V. 2.51

HOT Z
Notes

1710 Oliver Ave
San Diego, CA 92109

GETTING STARTED

The enclosed tape contains:

- o Version 2.5 of HOT Z for the 2068 computer. This is the bank-switching version that runs in any of the three memory banks built into the 2068. It supports all of the common printer interfaces and is PROMmable. It is recorded as a data tape and must be loaded with LOAD "" CODE

- o Version 1.8 of HOT Z-2068. This is a smaller HOME bank (ordinary RAM) version with a slightly smaller command set than v. 2.5. It supports only the 2040 printer and cannot be PROMmed. It provides access to all memory below address BC00H.

- o Version 1.9 of HOT Z-2068. This is a clone of version 1.8 that provides access to all HOME RAM above A900H. Both v. 1.8 and 1.9 include a BASIC loader/saver program, so load them with LOAD "" and wait until the double load finishes and HOT Z autostarts.

These notes deal primarily with v. 2.5. The other versions are operationally the same, except for the lack of a few commands and the memory bank management. You will only need the smaller versions if you refuse to enlarge the memory of your 2068 and require access to addresses used by v. 2.5 (8000 - DFFFH).

After LOAD "" CODE, you can cold-start HOT Z in HOME bank with RANDOMIZE USR 32777. You can then burn the code to EPROM, transfer it to another bank on a non-volatile RAM board, or exit back to BASIC and save the code to disk. (Code starts at 32768 and is 24576 bytes long.)

When HOT Z is cold-started, you must make a choice of printer interface. Select the 2040 if you have no printer attached. Your printer selection will start HOT Z with disassembly of the first few addresses of the HOME ROM. Enter any hex address to move the disassembly there; 8000 will find the beginning of HOT Z.

HOT Z commands are issued with the various Symbol- and Cap-Shifted keys. These keys are generally referred to both by their BASIC equivalents (e.g. PEEK or ATN or OR) and by the actual keying sequence, where CSS means Caps & Symbol Shift and release, and SS means Symbol-Shift and hold during the following keypress. Thus FLASH and CSS-SS-V are equivalent, as are <= and SS-Q.

HOT Z deals with blocks of memory by marking them out with a cursor and an entered address known as END. The value of END is displayed whenever the cursor is turned on. The cursor can be turned on with either SS-E (>=) or SS-A (STOP). The value of END will appear at the end of the second screen line. It can be changed by giving the TO (SS-F) command and then typing the desired address in hex. Turn off the cursor by hitting ENTER. (If you get tangled up in the mnemonics line through mistyping, escape from that first by hitting the semicolon, then ENTER.)

So to move v. 2.5 to either EPROM, NV-RAM, or a backup tape, first turn on the cursor and set END to DFFF. This marks out the 24K block occupied by HOT Z. To save that block to tape, give the CSS-S command (RESTORE), type HOT Z for the name, hit ENTER and proceed as in BASIC. Such tapes are loadable from BASIC as CODE tapes.

HOT Z on a NV-RAM Board

To transfer HOT Z to a NV-RAM board, set the board's bank switch to either Dock or Exrom (Dock will autoboot, Exrom won't.) and the protect switch to WR (write). Then, with the cursor on, give the transfer command (CSS-T). The query S/D Banks? will appear on the top line, requesting the source and destination banks of the code to be moved. The Home bank, here the source, is bank FF; the Dock is bank 00 and Exrom is bank FE, so enter either FF00 or FFFE (no spaces) depending on how you have set the switch on your NV-RAM board. You will immediately be asked for Dest, which is the destination address to which the code is to be moved. In this case you want the code to remain at 8000, so type that. After you have typed the last zero, you have the choice of aborting the operation (if you've made an entry error) by hitting the space bar, or of executing the operation by hitting ENTER. There will be a small flash on the screen and the code will be moved. To look at memory in another bank, give the CSS-G command with the cursor off and type in the desired bank number (FF, FE, or 00). This command switches on memory above 32K; see the notes on the variable DISB for switching low memory banks.

After you have transferred HOT Z, move the protect switch of your NV-RAM board back to the PR position. If you have moved HOT Z to the Dock, you have only to switch your 2068 off and on again to autostart HOT Z with a mostly clear Home bank. To start HOT Z in Exrom, go to BASIC with SS-Q and execute the following command line:

```
          OUT 255,128:OUT 244,240:RANDOMIZE USR 32776
```

The Exrom version is most useful if you also have some means of using the Dock bank for other code. There is also least chance of conflict with any other devices attached to your 2068. Running in Exrom is probably the only suitable way to use HOT Z with the RAMEX disk system, for example.

HOT Z on EPROM

If you have an Oliger burner board, HOT Z will burn itself to EPROM for you. You will have to set the END value according to whether you are burning 2764's or 27128's. The 8K segments are 8000 - 9FFF, A000 - BFFF, and C000 - DFFF; just span a pair for 27128's. (With a pair of '128's, you can have 8K of your own code above E000.) Set cursor and END for the size of chip to be burned, give the FLASH command (CSS-SS-V), select the type of chip you have in your burner from the menu, enter 0000 for DEST, hit ENTER, turn on the burner power, strike a key, and wait for the beep and Power Off message. Turn the power off, strike a key, and remove the finished EPROM. Install the two or three chips in a cartridge board mapped to 8000, and HOT Z will autoboot.

HOT Z on Disk

With the Aerco disk system, you may want to move HOT Z to Dock RAM as described for the NV board, then exit to BASIC (SS-Q) and save the Dock version to disk as an .aro file. You can also use it on a NV-RAM board in Exrom bank, which is the necessary way for the Ramex system (or use v. 1.9). I have heard from users who are running it from disk with the Timex Portuguese system, but I have no information as to whether that system allows full bank access.

HOT Z in Home RAM

The only limitation you must observe when running in a single bank of RAM is that you do not overwrite the HOT Z code. If your test program gets off the leash and runs wild, it may destroy the resident copy of HOT Z and cause a crash, as will errant memory transfers. This mode is useful from reading code that is already on cartridge, but note that the only bank that the 2068 can save to tape is Home RAM.

INTRODUCTION

HOT Z combines a line-by-line assembler, a labelling disassembler, a single-stepper and a simple editor. The purpose of HOT Z is to give you a reasonable degree of direct control of your computer, as well as to assist you in writing assembly-language programs to extend your control.

HOT Z requires some knowledge of the hexadecimal (hex) number system, which uses the characters 0-9 and A-F as its 16 digits. These instructions were written with the assumption that you know the fundamentals of Z80 machine code, for which there are numerous books on the market. If you are learning, then use HOT Z as a blackboard to work out the exercises.

This section provides an introductory tour of HOT Z. The experienced and the adventurous among you will want to plunge right in. If so, arm yourself with the short command lists and and try your luck. Details of the various commands are available in the later sections of these notes. Version 2.5 has somewhat abbreviated on-screen help, which can be reached with CSS-H from any of the three modes.

HOT Z comes up showing the first screen "page" of disassembled ROM. Down the left side of the screen, you will see the memory-address column, to which everything in HOT Z is keyed. These addresses are in hexadecimal and in the format accepted as input by the program. In other words, all addresses are four hex digits and include leading zeroes but no identifying symbols either before or after. The format is always there for you to consult as you make entries to HOT Z. Addresses run from 0000 to FFFF.

The second column of the disassembly display lists the contents of each memory byte, again in hexadecimal, two digits per byte, packed together with no spaces between. These numbers occur strictly in the order they occur in memory, which is not necessarily an easy order for reading. This column is raw data, as it were, against which any "interpretation" can be checked. Z80 instructions can be from one to four bytes in length. A HOT Z routine gets the length of any instruction and parses the bytes into instruction- length clusters, but it cannot decide whether those bytes hold true Z80 code, as here, or simply numbers used as data. That decision in the end is up to the reader. On this first page of ROM, the first two instructions are one byte long, the third three, etc.

The next column, the NAME column, will hold user-entered labels for the corresponding address, along with a few labels provided in a permanent file on your original tape. After you have annotated a program with these labels, you can SAVE a NAME file separately from HOT Z, to be loaded again with whatever program the labels pertain to.

The fourth column presents those particles of electronic poetry known as assembly mnemonics. Relative jumps (JR's) are listed, as in the sixth line, with their destination address (or NAME) rather than the single displacement byte with which they are coded. System variables for the ROM are listed by an abbreviated name, as in lines 4 and 5.

The first four instructions turn off the keyboard interrupt, set A to zero load DE to count 64K of memory, and jump to the initialization routine. The rest of the screen is taken up by RST routines. RST 10 prints the character whose code is in A, RST 08 handles BASIC error reports, RST 18 and 20 help with interpreting BASIC, and RST 28 is the entry to floating-point operations, which are a separate sub-language in the 2068. RST 08 and 28 are always followed by one or more (for 28) bytes that serve as data rather than as machine code. The meaning of such bytes is listed in the mnemonics column if you have the floating-point interpreter switched on.

The current HOT Z display is referred to in these notes as READ mode or disassembly. The commands in this mode are mainly for moving the display around to give access to different parts of memory. The page flip, for example, is the SPACE bar; hit it to continue the disassembly with the instruction following the one at the bottom of the screen. For distant moves, you can enter a four-digit hex address to the ADDR cursor at the upper-left screen corner. For example, try 0031 to see the initialization routine.

During address entry, you can backspace to correct an error by using the DELETE key, which will back up the cursor one space. DELETE doesn't blank out the entry and that you can't back out of the whole entry routine that way. To back out, use the ENTER key, which works as an escape key in this situation. You must type in all four hex digits of an address or all four characters of a NAME (label). ENTER is not needed after the last hex address digit.

The HOT Z keyboard responds almost identically to way it responds in BASIC. HOT Z gives a different tone feedback (You can alter that by changing pip_.) and gives the tone for CAPS LOCK and the SYMBOL-SHIFT/CAPS-SHIFT (CSS) combination as well. CAPS LOCK is initially set. Lower-case a through f are not recognized as hex digits, so if you shift to lower case to enter a label, be sure to shift back before entering hex or Z80 mnemonics. The lower-case mode is indicated by cursor flashing and bright rather than just flashing. All the shift-key entry combinations are the same as in BASIC, except that the K-cursor state is not used by HOT Z, so the keyword legends on the keys themselves are not available.

In READ mode, you can also get to a named routine by entering the four letters of an assigned NAME. Try KEYB. You will see that the NAMEs appear in both the NAME column (referring to the current address) and in the mnemonics column (referring to the target address of CALLs or jumps).

In general, you can use a NAME in the file as a proxy for its address in the READ, Assembly-Edit, or One-Step modes of operation.

If you did not do so before loading, set the screen to your favorite color combination using the BORDER (on CSS-SS-BORDER, i.e. the BRIGHT key), PAPER, and INK commands. They work essentially as in BASIC, except that the color comes up right away.

Try keying SS-G (THEN) from READ mode. This is the display switch, and successive strokes of the the same key will take you back and forth between the data and the disassembly displays. The data display is for examining those parts of memory that are used as files of data rather than for Z80 code. The first and second columns contain the single address and its content in hex, values that are reflected in decimal in columns four and five. (Use it as a conversion table.) The far-right column gives the CHR$ of the contents of the address and will turn up any BASIC programming or message files. Enter, for example, the address 0227 to see the keyboard file. Switch back to disassembly while you're still looking at the keyboard file for a taste of what disassembled data (sometimes called nonsense) looks like. It's up to you to distinguish sense from nonsense when reading a strange program; the display switch is there to help you do it.

The NAME column in the data display functions differently from the column with the same heading in the disassembly. The NAMEs in the data display are those that correspond to any two successive bytes, taken in lo-hi order, in the second column. (The disassembly displays NAMEs assigned to the addresses in the first column.) Some NAMEs in the data display can crop up by chance; for example, two NAMEs immediately together mean that at least one is spurious.

Use the CSS-T command in READ mode to go to the beginning of the NAME file. The NAME file grows downward like a stack, which it is not, as you add new NAMEs to memory addresses. Turn on the data display to see the structure of the NAME file. Each NAME takes six bytes; the first two hold the address to which the NAME is assigned, hence the listing in the NAME column, and the next four hold the NAME itself, which shows in the CHR$ column. Other odd CHR$ symbols will appear at random for some of the address bytes, signifying nothing.

The data display is also useful for looking at BASIC programs to see the real structure of BASIC code.

You can enter decimal addresses to the ADDR cursor, but these must be prefixed by the OR (SS-U) command. Try it, and check the conversion with the data display. If you enter a decimal address of less than five digits, then you have to press ENTER to tell HOT Z that you've finished. If you enter a decimal higher than 64K, the program will subtract 64K and give you what's left.

Now get into disassembly and go to 3B2E, which is where the ROM begins the BASIC function LN. Hit CSS-O (PEEK) to turn on the floating-point interpreter. The first instruction after the RST 28 restacks the number on the top of the calculator stack in full five-byte form (in case it is a short integer); the number is then duplicated on the stack and tested for being positive non-zero; if it is, a jump is made to 3B37; otherwise, execution proceeds to end the floating-point code and fall into the trap for error A. At 3B37, we have an example of floating point code that is embedded and not preceded by an RST 28 because of the jump. To get the correct interpretation, enter 3B37 to the ADDR cursor, then use the switch command on the CSS-I (CODE) key.

At 3B35 you will see a rendition of a BASIC error report after RST 08, in this case for a zero or negative argument to the logarithm. Occasionally, you will encounter a CF as data rather than RST 08, in which case the error number may be invalid and left blank.

The last display on the tour is the Z80 register display or Single-Stepper. This mode can be entered by using the STEP (SS-D) command from the disassembly.

The register display occupies the top three quarters of the screen. The left column lists the various Z80 registers; please refer to a good Z80 reference book if you need an explanation of the register names. The exchange flags are listed as EXFLAGS.

The second column lists the hex values of the registers' contents. Values for the accumulator (A) are listed at the left of the column to remind you that A is the high half of the AF register pair, along with H, D and B. The third column either converts the second column value to signed-decimal according to the two's complement convention, or, if the second column holds an address that has been NAMEd, then that NAME is listed in the third column. The fourth column, headed by the open parentheses, gives the hex value of the byte contained in the address formed by the register-pair values. (E.g., across from HL you will find the byte (HL).) The right column gives the CHR$ of the byte in the fourth column (for the register pairs) or of the byte in A.

The box below the one containing the exchange registers holds details on the one-step user's stack and the state of the flags registers. The user's stack is separate from the main machine stack so that the system can absorb a few stack errors without crashing the program. The top four pairs of bytes on the user's stack are shown at the right, along with the NAMEs for any addresses they might hold, so that you can check to see whether your test routines leave anything behind. The main flags are listed below the exchange flags for easier visual association with the conditionals in the program steps below. Standard conditional mnemonics are given for the four programmers' bits.

The cursor at the left in line 18 (which is bright) marks the address of the next step set up to be executed by the single-stepper. You can enter any address into that cursor just as you would in READ mode, or you may also use a NAME. The ENTER key still serves as an escape during address or NAME entry, but it has another more important function as well, which is to run the next single step.

If it's not already there, enter 053A to the NEXT slot, and then notice the contents of the A and C registers just before and after you press the ENTER. This is a fairly safe area and you can experiment with a few more steps. (The things you must be careful about are loading into some system variables, either ROM's or HOT Z's, and some flag sets. The SPACE key allows you to skip the step at NEXT. The top line of Z80 instructions represents the previous step executed, and the three steps following the one in NEXT are those that will be reached if there is no branching. A branched-to step appears directly in the NEXT slot; a skipped step disappears from the display.

For faster debugging, you can set breakpoints (AT and OR commands) and use the SS-G (THEN) command to step through the code as far as the first breakpoint encountered. Two breakpoints are provided so that your can cover both sides of a conditional branch. You must take care to set breakpoint addresses that the code will actually encounter, since stopping depends on finding a breakpoint exactly. The BREAK key will stop the CSS-G command if used quickly enough. You can display the current breakpoints with the SS-Y (AND) command.

Breakpoints are only checked for in your main code line, not during any subroutines (CALLs or RSTs). This may not be ideal for all your tests. If you want to set breakpoints within your subroutines, then change the RTBP (D0ED) routine as follows: the second instruction (D0F0) should CALL STEP (CD71D2) and the second last instruction (D10A) should CALL STE2 (CD40D5). If you make these changes, then don't use both the window and code with RST 10s that you run to breakpoint.

Learners might consider mastering the use of the Single-Step first and then using it to see how the various instructions and a few resident routines work. A lot of bugs can be avoided by testing every routine you write with this device. You can also create a special display screen that will show the display of your test routine and alternate with the register display. See the section on the Single Step Window for details.

Hit SS-Q (Quit) to get back to the main READ display. You will arrive at a screen page that starts with the address that was in the NEXT slot of the Single-Stepper. If you spot an error coming up at the bottom of the Single-Step display, you can quit the display, EDIT the error on the disassembly display, and get back to where you were in the Single-Step by using the STEP command from READ mode.

You can also go directly to assembly mode within the Single Step display to make minor changes to upcoming code. The CSS-A key will give you a cursor at the head of the mnemonics column and let you make changes without exiting Single Step. You are effectively in the edit mode with a return address to Single Step on the stack. Consequently, all of the edit commands are available to you, but you must make judicious use of them. It would not be wise, for example, to invoke the Single Step while editing under the Single Step.

A number of operations may redo the screen to the EDIT mode or otherwise damage the register display. However, the Single Step screen will reestablish itself as soon as you exit the EDIT mode by hitting ENTER.

Operations that move to a different address in edit will not change the current address in the NEXT slot. That will be preserved just as if you had left the Single Step and then come back to it. Moving the cursor out of the disassembly area into the register display is usually prevented and not advised.

SOME ESSENTIALS

DISASSEMBLER FEATURES

The HOT Z disassembler has been specially programmed for the Sinclair ROM to take account of the system variables, the BASIC error reports, and the floating- point operations, which make up the Sinclair 'calculator language'.

Abbreviations of system variable names are included in the permanent NAME file that loads with the program. The HOT Z disassembler always uses the name for a system variable whether it is referred to by absolute address (e.g. 5C72) or by a displacement from IY (IY+38). However, if you want the IY form from the assembler, you must write it out, since the assembler will always substitute an address (two bytes) for an entered NAME.

When an RST 08 occurs, the following byte is not Z80 code but is used as data to generate the BASIC error report. HOT Z reads these bytes as ERROR 9, etc., rather than generating Z80 mnemonics for them. If you are running the disassembler over a block of data, you may encounter a CF (hex for RST 08) followed by a byte that would be out of the range of the error reports. In that case, the error number is not printed.

An RST 28 is the ZX ROM's entry into the floating-point language, which can be disassembled by HOT Z. You can switch the f-p language interpreter on or off with the CSS-O (PEEK) command in READ. The default on start up is off. If you want to know what is going on in the floating-point routines, then consult appendix A of these notes.

PRINTERS

The Oliger, Aerco, Tasman, or A&J printer interfaces are supported in addition to the 2040. You are asked to choose which interface at boot up. If you use a Centronics interface and you find that your printer double spaces HOT Z's output, then you can change the code in RAM at 5DE3, which sends a carriage return and line feed at the end of each line, to send just the line feed.

If you burn your own EPROMs or run HOT Z in RAM, then make the above change in the template code at 8942.

COLOR

You may also want to change the color byte at 800A. Set the colors you prefer either from BASIC or with the HOT Z commands, and then look at the attribute file (5800-5AFF) and install the predominant byte you find there at 800A.

CONFLICTS

HOT Z keeps its error fielder at 5C2F in the streams area of the system variables. If this interfers with any of your peripherals, then change 5C2F to 6824 at 808F and 830C. Changes can be made in a running version and will take effect on next boot up. (Color is immediate.)

HOT Z takes over the printer channel pointer and does not restore it. If you move back and forth between BASIC and HOT Z and expect to print from both, then you will need to restore the address of your printer driver at 684F (26703).

If you use all three banks of memory, then you must keep account of the value in port 255. It is possible for that port to hold 128 even when no EXROM chunks are enabled. (Port 244 = 0.) If the value of port 255 is undetermined, then you won't know whether you are enabling Dock or EXROM chunks with port 244.

THE DISASSEMBLY BANK VARIABLE (DISB)

In addition to the bankswitch command (CSS-G), the variable DISB (disassembly bank) can be manipulated directly by the user to control what you see with the disassembly and what memory you change with HZ commands. DISB is a two-byte variable that is actually a bank-chunk spec; the high byte is the bank (FE = EXROM, FF = HOME, 00 = DOCK) and the low byte is the active-low chunk-enable byte (00 enables all chunks, FE enables chunk 0, 7F enables chunk 7, etc.) The default on start up is FF00, which is all chunks of the HOME bank.

Most values can be written in directly, but there are a few combinations that hang the machine. All zeroes, for example, mean enable the dock everywhere, which locks out the stack, as does any combination of bank and chunk spec that turns off chunk 3 with the stack in it.

Valid combinations of bytes for DISB will depend on what you have connected to the 2068. If you can hook up a chunk 0 in some bank, then you should have an interrupt fielder at 0038 as a minimum before you enable such a bank without a DI. You can copy out the code from 0038 to 0048 in the EXROM if you need a fielder. Chunk 2 contains the system variables and the HOT Z RAM-res code, and you will have to come up with a smart routine to make use of that chunk. Finally, chunk 3, from 6000 to 7FFF, contains the stack, and that must be moved to an active RAM chunk before you can switch out the Home RAM chunk 7.

Awkward values for DISB can generally be avoided by replacing them backwards (high byte first) or by using the Transfer command to move two bytes into DISB together.

RAM USE

HOT Z-AROS has its variables and buffer area in RAM at 5F60-5FFF. This could ultimately get in the way of the Syscon parameter table for memory banks and intelligent devices, but there is room for four or five, which should do for the near future. HOT Z uses a RAM-resident block of code, which is presently located between 5D00 and 5DFF. This could cause conflict with other devices or programs that use the same area. HOT Z does not use the 5E00-5EFF area. Your workspace in RAM runs from 50 bytes above STKEND to FFFF and of course any other banks not occupied by HOT Z.

HOOK COMMANDS

For use with EPROMs the PI and the TAB keys can be hooked to your routines in RAM to turn them into HOT Z commands. All you do is write the address if your routine at the appropriate address. Those are as follows:

READ:	PI	5F90
	TAB	5F92
STEP:	PI	5F94
	TAB	5F96
EDIT:	PI	5F98
	TAB	5F9A

It will not be possible to write an address to the command file, if the command file is in EPROM. The routine that you hook up must be in normally enabled RAM, which is to say RAM below 8000H. You can enable and call into high RAM with CALL 5D07, CALL YOUR_ROUTINE, JP 5D00.

WRITING AND EDITING Z80 CODE

The READ mode is a essentially passive, allowing you to page through the memory and examine its contents. The WRITE or EDIT modes are there to let you make changes in the memory content, provided that memory is RAM.

There are three WRITE/EDIT modes. With the disassembly display, you can press CSS-A (STOP) and a cursor will appear at the top line of the edge of the right column. This is the Assembly mode. Once you turn on the cursor, you change the entire command system of HOT Z. The commands available to you with the cursor on are listed as the EDIT-mode commands on the command lists. Hitting ENTER with the cursor in its "home" column will quit the WRITE mode and return you to READ, where you can readjust the screen to another part of memory.

In addition to the command set, the up and down cursor controls allow you to move the cursor to a given line or to scroll the display page one line up or down by moving the cursor up from its top position or down from its lowest position. Up scrolling is automatic when you ENTER a line that is third from the screen bottom.

You may also enter a new Z80 instruction to replace the one listed on the cursor line. Just start typing and the existing line will disappear. As you type, the delete key and the left and right cursor controls will function as you expect them to. If the cursor is over the top of a character, your next keystroke will replace that character. If you want to insert a character, press the EDIT key and a space will be created at the cursor position, with all characters to the right of the cursor being shifted one space right. The rightmost character in the line (usually a blank) is destroyed by this insert command. You cannot jump to another line with the up or down cursor command while you are in the middle of editing a given line.

When you have entered the intended Z80 instruction, hit the ENTER key to put the proper code into memory. If your entry is in the proper format, the cursor will return to the left edge of the column and move one line down, ready to edit the next line. If the cursor stays put in the line you are working on, then it indicates a format error in the mnemonic entry.

HOT Z follows the format of the mnemonics listed in the Zilog Z80 technical manual. This format is the same as that listed with the character set in your computer's instruction manual, with the following exceptions: the RST's are followed by a hex byte (08,10,18,20,28,30,38) rather than decimal and the OUT (N),A and IN A,(N) use the parentheses shown here. (N is always a two-digit hex byte.) The open parenthesis is always preceded by either a space or a comma, and spaces are always important.

When HOT Z fails to accepts your entry, it locates the line cursor at the first position that does not match its template for a proper instruction. Sometimes, however, as with an omitted space or an unassigned label, the cursor may appear earlier than your particular format error. (For example, it will flag the first letter of a label even if only the fourth letter is "wrong".)

If you get stuck and can't get HOT Z to accept what you've entered, you can abandon ship and restore the original mnemonic by hitting the semicolon (;). Your recourse then is to look elsewhere in the disassembly for the format of the instruction you have been trying to enter, or to look up the hex code for that instruction and to enter that in the hex column (See below.) to discover how HOT Z lists the mnemonic.

If you try to back out of a line with the cursor-left key, HOT Z will act as if you have tried to ENTER the line. If you write all the way to the end of the line an ENTER will also be automatically appended. This occurs with some of the IY+N instructions, which just fit in the alloted space.

You can use a preassigned NAME in an instruction anywhere that a 16-bit (four hex digits) number occurs. For example, LD HL,(rmtp) is equivalent to LD HL,(5C82). You must give a NAME to a particular address (CSS-N or INKEY$ command in WRITE) before you attempt to use it in an instruction.

Upper/Lower Case

Since HZ does not recognize lower case for hex input nor the main part of a mnemonic, it can be inconvenient or even puzzling to be in that shift state on an RGB monitor with no bright cursor to indicate what is happening. There are a few automatic turn-offs of the lower-case state: after entering a new NAME, after entering an assembly line, and on turning on Hexedit. The shift state does persist if you enter a lower case NAME to the top line cursor in READ mode; this causes it to fail to recognize addresses like 5c77 until you retoggle the caps lock key.

Jump Instructions

Relative jumps (JRs and DJNZ) are normally entered with the destination address or NAME. However, for the JRs only (not DJNZ) a second form is available for short forward jumps where you haven't yet assigned a NAME but know how far forward you want to jump. JR +5 will jump ahead over five bytes. The plus sign is required and the displacement is in decimal with a range from 0 to 127. Backward jumps are not catered for in this way; it is easier to look back for the address you want to get to.

Provided you do not want one of the last four conditional expressions (M, P, PO, or.PE), you can use relative jumps all the time, and if the destination address is too far away HOT Z will convert your JRs to JPs (absolute jumps) rather than report an error. The reverse is not true: if you enter a very short absolute jump, HOT Z will take your word for it. This conversion works well for entry of new code, but you must beware when editing in the middle of an existing routine, because if a two-byte JR is edited and becomes a three-byte JP, then the first byte of the following instruction will be overwritten.

Pseudo-Ops

There is no ORG command because you are doing the ORG yourself with HOT Z. However, direct data entry is possible in the assembly-edit mode through use of the DB pseudo-op. DB may be followed by a quoted string (DB "ABCDE") or by an even number of hex digits (DB 090F 0D3A). Spaces are ignored in reading the hex digits, except for the required space after the DB. Each pair of hex digits is read as one byte, and a single digit left over will be ignored. You can write a string or series of digits all the way to the end of the line.

When you hit the end, HOT Z will add the quote if necessary and enter the line. Upon entry, the editor enters one character (for a string in quotes) or two hex digits per byte starting with the cursor address for as many bytes as it takes, then resets the screen layout so the next cursor address is at the top of the screen. The reason for this is that the data you have entered would be disassembled by HOT Z, producing a nonsensical listing. You can look back with the data display to assure yourself that what you have entered is indeed there.

The DB is simply a means of entering data without leaving the assembly-edit mode. You should still assign NAMEs to your strings or variables and use them in referencing the data. The insert command is recommended when you enter data into an existing code block.

If you want to use the RELOCATE command (described below), then you should not mingle small blocks of code and data. Keep them in large blocks and keep track of what is where.

In addition to string entry with DB, you may also enter quoted non-inverse characters for direct eight-bit register loads or for direct arithmetic/logic operations. LD A,"A" will assemble as LD A,41 and CP "Z" as CP 5A. Sixteen-bit (double) register loads are not treated in this way.

Hex Edit Modes

Hit the >= key with the disassembly display to get into the main hex edit mode. The "home" column for the cursor in this case is between the address and hexcode columns at the left of your screen. Cursor controls work as with the assembly editor.

To change the hex content of memory, you may either move the cursor over with the cursor-right key or retype the line, using the keys from 0 to F. With the disassembly display, each line holds the correct number of bytes for a single Z80 instruction. If you write a one-byte instruction, the cursor will jump to the next line immediately; for multi-byte instructions, the cursor waits on the line until the required number of bytes have been entered, then jumps automatically.

The purpose of this feature is to allow you to copy hex listings from printouts or magazines. You can just type away without worrying about hitting ENTER at every line, and the screen will scroll along with your entries.

With the edit mode, what you see in the hex column is what you get when you make an entry, byte for byte. Edit does not use NAMEs and you have to calculate the displacements for any relative jumps you enter.

All of the EDIT-mode commands are available with the hex-edit cursor on screen. There is, however, no character insert while you are editing a line, and the escape key in the middle of a line is ENTER rather than semicolon. If you need to change the first byte of a line after you have started editing it, you should escape by hitting ENTER and start over.

You can hit the SS-G (THEN, display switch) key either before or after you have gone to the hex-edit mode in order to obtain the data-edit mode. This mode lets you change one byte at a time by writing a new value over the top. This is the mode that you would use for entering hex data files, addresses and the like. (Use the DB command from the assembly mode for entering text files.) All write commands are available from this mode as well, except the NAME (CSS-N) command functions differently than it does with the disassembly display. CSS-N will no longer assign a new NAME, but can be used to write a preassigned NAME to the NAME column, and the address to which that NAME belongs will then appear at the cursor address and the byte following. The intended use is for creating address files (jump tables).

Inserting and Deleting Lines (All EDIT Modes)

What happens when you press ENTER after writing an instruction is that HOT Z reads the address of the line you are working on, looks up the the numeric code of the instruction, and enters that code into as many bytes as it takes. Then control goes back to the disassembler, which reads back your code into Z80 mnemonics and revises the screen page accordingly. An important consequence of this is that when you are editing an existing block of code you must be careful not to overwrite more lines than you intend to (by entering a four-byte instruction over a two-byte instruction, say) and to watch out for new instructions that crop up when you overwrite a long instruction with a short one (one-byte over a three-byte instruction, for example).

If you don't know the byte length of Z80 instructions, the way around the above problem is to use the line-insert (EDIT) and line-delete (DELETE) commands whenever you are editing an existing block of code.

When you insert or delete a line, a block of code is moved either to make room or to close up the empty space. One end of that block of code is determined by the cursor; the other end must be determined by you before you start your editing session. Whenever the WRITE cursor is on, a variable called END is displayed in the upper right corner of your screen. END marks the other end of the active memory block for an insertion or a deletion or indeed for any block operation, such as a clear, a fill, a SAVE, or a transfer. END is set with the TO key (as in TO the END) followed by four hex digits or a NAME. On some types of entry errors, you may be asked twice for the proper value.

You should set END whenever you begin an editing session. For the insert-line and delete-line commands, END must be within 256 bytes of the cursor address, or else you will be asked to enter a new value of END when you give the insert or delete command. At that point, HOT Z will accept any value you enter for END and perform the operation. The purpose of this behavior is to catch those times when you have forgotten to set END, and to save you from destroying valuable code.

There are three separate commands to set END, just to make it easy. The TO key will work in either EDIT or READ modes, or you can use the OR (SS-I) key in EDIT mode to pass the address at the cursor directly to END. END is generally always on screen when you need to know it.

For insertions and deletions, END can be either above or below the cursor address. The "usual" value would be for END to point to an address higher than the cursor address, in which case an insertion would push all values to higher addresses to make room for the new instruction. For example, if you insert a two-byte instruction at 8C10 with END set to 8C80, then all instructions from 8C10 will be moved two bytes higher until 8C7E, which will go into 8C80, and the original contents of 8C7F and 8C80 will be destroyed. A deletion of a two-byte instruction would move all instructions to lower addresses, and the contents of 8C7F and 8C80 would be duplicated in 8C7D and 8C7E.

On the other hand, if the address in END is lower than the cursor address, then an insertion will leave the following addresses undisturbed but will push the contents of preceding addresses to lower addresses as far as END. For example, with END set to 8C00 and the cursor at 8C10, insertion of a three-byte instruction would destroy the contents of 8C00, 8C01 and 8C02 by overwriting them with the contents of 8C03, 8C04 and 8C05, respectively. Analogously, a deletion would duplicate the first three (or N) bytes in the next three. The insertion itself will in this case go into the address preceding the cursor address. This feature is useful when you are editing in a constricted memory block with blanks that may be either above or below.

After insertions or deletions, the cursor position may have to be adjusted for your next entry. (The preceding discussion uses "above" and "below" to refer to numerical values of addresses, not to screen position, where addresses get higher as you go down the screen.)

When a NAME is assigned within a block where you are inserting or deleting lines, the NAME will move with the instruction to which it is assigned. The displacement assigned to relative jumps is not adjusted, so JR TARG may read JR 8C22 after an insertion that pushes TARG from 8C22 to 8C23. Be sure and label all JR destinations and then check that the labels are still correct after an editing session. If you use labels all the time, then an error will stand out clearly.

When you are editing the data display, all insertions and deletions affect one byte at a time.

Using EDIT Commands

Many of the EDIT commands affect a block of memory and require that the END variable be set first to a proper value. Use the TO key to set it. Aside from its use for insertions and deletions of lines, END is generally set to denote the end of a block of code, whereas the cursor marks the beginning. If END is less than the cursor address, the block is generally taken to be null, though sometimes the operation will still affect the very first byte. Most operations include the END address; the exceptions are SAVE and LOAD, which finish one byte before. (This makes it effectively impossible to LOAD or SAVE address FFFFH, since the next address is 0000, which is less than any cursor address.)

The block commands are LOAD, SAVE, FIND, TRANSFER, CLEAR, FILL, LLIST, READDRESS and RELOCATE, in addition to the line insert and delete described above. The simpler commands are SS-A and SS-E, which toggle the cursor across the screen between assembly-edit and hex-edit; SS-G, which toggles the display between disassembly and data and works only in hex-edit because you can't assemble data; CSS-N and CSS-X, which allow you to assign or delete a NAME at the cursor address; STEP, which takes you to the single stepper; and CSS-RUN, which transfers control to the program beginning at the cursor.

The cassette commands (LOAD, SAVE, VERIFY) allow you to move the contents of individual blocks of memory to and from tape in the CODE format. Such tapes will be loadable by the corresponding BASIC command if you calculate the length (END - cursor address) and work out the decimal values. Similarly, CODE-format tapes made in BASIC will load with HOT Z when you have made the numeric conversions to hexadecimal. The BREAK key works to interrupt any of the cassette functions. Error reports will appear on screen with a BEEP, and the system will wait for a keystroke before accepting any further commands.

Cassette functions all require tape names, which are entered without quotes after you give the command and before you press ENTER. Maximum length for such tape names is the standard 10 characters. An incorrect loading space (END minus cursor address) for the tape in question will result in a tape loading error. If you get such an error, you can inspect C5BF and following addresses with the data display: the length you enter is at C5BF +0C, the length read from the tape at C5BF + 1C. Then correct your setting of END.

The TRANSFER command allows you to move the contents of one block of memory to another block. The first thing to do is to make sure that your destination block will hold the source block without overwriting something you want to keep. You have the option of copying just the code with CSS-T (RND) or of copying the code and moving the NAMEs assigned to it as well with CSS-SS-T (MERGE). The original of the code will not be erased by this command. You can copy from ROM but of course not into it. You can only move NAMEs if you have the file in RAM.

To use the transfer command, set END and hit the appropriate command keys. This will bring up S/D Banks? (Source /Destination) in the top line. For normal use in Home Bank, just respond by hitting ENTER. For interbank transfers, consult the first section of these notes. After you respond to the Banks? request, a DEST cursor will come up at the upper left, which asks you for the destination address of the block. HOT Z will wait for you to hit ENTER after that address, and if you change your mind or find you've entered it incorrectly you can bail out by hitting the SPACE key instead of ENTER. After the command has executed, the display will move to the address you gave to DEST.

The FIND command has a similar protocol to that of transfer, but it works only in the bank that is on display via DISB. In this case, set the cursor to the beginning of a block of memory for which you want to find a match. Set END to the last byte of your template. Hit CSS-F (SGN). An address cursor labelled LOOK will come up at the upper left. Enter the address at which the search should begin; hit ENTER to proceed or SPACE to back out. HOT Z will search 32K (8000H) bytes for a match to the memory from cursor to END; if a match is found, the display moves to it; if there is no match, the display remains at your template in READ mode. If you find one match and want to search for another, set the cursor again, move the cursor down a line or two so it doesn't point to the beginning of the found match, and use the CSS-G (ABS) command. If a second match is found, the display will move to it; if not, the display stays put. (NOTE: If you are searching for a block of 8 zeroes, say, and you find a block of 12, then to continue the search you should move the cursor down so that there are 7 zeroes or less below it, or else you will find the same string all over again.

The CLEAR command (ERASE) will put zeroes in all bytes from cursor to END. The FILL command first asks you for a keystroke and then fills the block with the code for the character assigned to that key. (If you clear or fill a block of HOT Z or the stack, you are likely to crash.) To fill with a value not available from the keyboard, write that value to the HOT Z variable FILC, then use the CLEAR (not FILL) command.

The LLIST command in WRITE will send the contents of the screen, starting with the cursor line, to your 2040 printer. Printing will continue, interrupted by page flips of the display, until the line just before the END address. If you forget to set END, you can BREAK to save paper.

There is also a hex-arithmetic command, which, though not a block command, uses both the cursor address and END. The command is READ, and the result is the hex sum and difference (END minus cursor address) of the two values, which are displayed in the command (top) line.

The Readdress (for jump tables and NAME files) and Relocate (for programs) commands are described in a later section of these notes, due to their complexity.

A detailed description of all the HOT Z commands is also included as a later section intended for occasional reference. Other sections will give you details on naming and NAME files, the floating-point language interpreter, and the program relocator.

HOT Z's Flags

HOT Z uses the BASIC system variable STRLEN as 16 bit-flags, so you could crash the system if you try to load that variable. The meaning of HOT Z's flags is that they are SET to indicate:

Bit	HZFG (IY+39)	STRLEN
0	Disassembly of RST 08	SP display
1	Disassembly of RST 28	RST 28 disassembly in progress
2	An insert	Unused
3	A NAME input	Unused
4	Data display	Unused
5	Hexedit not assembly	Assembly in STEP
6	F-p constants	Disassembly of APPX
7	Window in STEP	Transfer of NAMES

This use does not, to our knowledge, affect the operation of a co-resident BASIC program. However, if you run a BASIC program and intend to return to a resident HOT Z with a warm start, it is best to POKE 23666 and 23667 to 0.

THE COMMAND SET

Keying is described as CSS- for the Caps/Symbl-Shift combination before another keystroke and SS- for Symbl Shift pressed simultaneously with another key. Keys are referred to by any of the three rubrics on the keytop. Mnemonic associations are generally with the letter on the key: for example, Assembly is Symbol-Shift/A, the STOP key. There is a brief help screen that you can call up from READ or EDIT modes with CSS-H (SQR).

READ Mode

Key Description

QUIT TO BASIC

SS-Q Quit HOT Z for BASIC. HOT Z and the entire Dock bank are switched out so that BASIC sees only Home bank

COPY

CSS-COPY Copies the screen to the designated printer. Gives you headings and all. Consider using the LLIST command from an edit mode for no headings and variable length. LLIST is faster.

HEXEDIT

SS-E Sets the cursor to the top line and switches to the hex-edit mode. This command also works from assembly-edit mode without resetting the cursor line.

ASSEMBLE

SS-A Sets the cursor to the top line and switches to the assembly-edit mode. The same keystrokes will get you from hex-edit to assembly edit. This command works only when the disassembly display is on.

TOP NAME

CSS-T Move the display to the 'top' of the NAME file and switch to the data display. Use this command as preparation for SAVing a NAME file. (Turn on the cursor, set END, and SAVE.) If the file is still in EPROM and DISB is set to its default, you will see the corresponding memory space in RAM, which may be empty or hold something else.

NAME SWITCH

CSS-
SS-N
(OVER)

NAME file switch. If you are using only one file, the NAMEs are switched off or on. If you have two files in memory, the command will switch from one file to the other. Before switching, you must first write the start and end addresses of the new file at ALNA (lo-hi order). The end address is the first of two bytes of zeroes at the top end of the NAME file. To start a new file, set both addresses the same, pointing to two bytes of zeroes, then add names to the disassembly.

RESTART

CSS-R

Restarts HOT Z. Resets the stack to clear clutter. Resets register values in the single step and sets the EPROM-resident NAME file active.

MAKE REM

CSS-REM

Installs a 1 REM statement in BASIC at the value in the system variable prog (normally 7B16H). The REM will run to the value in END and will push other BASIC lines to higher memory.

BORDER-INK-PAPER

CSS-SS-BRIGHT BORDER color set. Follow with a color key.
-INK INK color set. Follow with a color key.
-PAPER PAPER color set. Follow with a color key.

STEP

SS-
STEP

Switch to single-stepper. The address in the NEXT and LAST slots will be last ones used there. Use this command to get back after an interruption. All old single-step register values are preserved.

DIS/DAT

SS-GOTO
(THEN)

The display switch from disassembly to data display or back again. The same command works with the hex-edit cursor on but not from assembly-edit.

SET END

SS-TO

Enter a value to the END variable, as in EDIT mode, but the value is not displayed

DECIMAL ADDRESS

SS-OR Indicates decimal address to follow. Clears away the ADDR cursor and waits for your entry. If the decimal address is less than five digits long, hit ENTER after the last.

SCROLL

SS-<> Sets the screen to a continuous SCROLL. BREAK will stop it. A toy.

SP ON

SS-AT Toggles on or off a display of the machine stack-pointer address in the upper right screen corner. The default is Off, because it isn't pretty, but you should turn it on when you are test running your own routines. There is a small amount of shock absorption in the HOT Z stack, but if you should see it changing, then look very carefully at what you are doing to the stack with the routine you are testing. Restarting HOT Z will reset the stack.

FP IN-OUT

CSS-O (PEEK) Switch the on-off state of the floating-point disassembler. If turned off, then the SS-I command will have no effect. If on, then every EF (RST 28) will switch to the floating-point disassembly and every 38H will switch off the floating-point disassembly. If you have a stray EF on screen while you are in an edit mode, you may get a messed up display when you enter code. If so, exit (ENTER) from edit mode, use this command, and go back into the active mode without fear. Default state is OFF.

FP INTERPRETER SWITCH

CSS-I (CODE) Floating-point interpreter switch. This is a flag switch (NOT an on-off switch) which switches interpretation of a byte from Z80 language to floating-point language. This command is necessary for certain embedded sections of floating-point code that are not preceded by an RST 28 but are jumped to from some other portion of floating-point code. This command will not function if the PEEK switch has been set to off. If it doesn't work, hit PEEK and try again.

BANK SWITCH

ABS (CSS-G) The bank switch. You can ask for FE, FF, or 00. The command is set to switch in only the top four chunks (32K). For chunks 0 to 3 of Dock or EXROM or for chunk mixtures you must still manipulate DISB; remember that Dock and EXROM don't mix because of port F4. "ro" means read only (ROM) and "rw" means read/write. "Forbidden" chunks can be reached via DISB after appropriate precautions (putting an interrupt fielder into Dock 0, moving the stack, avoiding system-variable references, etc.)

PRINTER CONTROL

FN Sends anything you have written in the printer buffer (at 5B00) to the Centronics port and your printer. Consult your printer manual and use it to send control codes to configure your margins and page size for HOT Z output. Stops at the first zero byte.

NAME FILE TRANSFER

INKEYS Sets up an empty NAME file at the top of RAM. Just give this command and add NAMEs as you choose. Then save your file from the address given by the CSS-T command to FFFF.

You need this command for almost any change of NAME file. If you have file on tape, use this command first, then load the tape, then set the file start and end at ALNA, then use the OVER command to set up the new file. If you want to pick up some part of the existing ROM file, then you will want to transfer interbank from bank 00 to FF. A handy way to do that is to use the "backwards" format of the transfer command. (Learn it straight up first.) To do that, set END to the beginning of the part of the NAME file you want to move (the low end). Set the cursor to the high end, the fourth letter of the last NAME you want, and set DEST to FFFD before you hit ENTER to execute. The display will show you the top of the new file, which you must then enter at ALNA before applying the OVER command.

DISPLAY MOVE

NOT Moves the display to the address in END.

PROGRAM BANK TRANSFER

VALS — Moves HOT Z from Dock to the EXROM bank, at the same addresses, if you have modified your 2068 to have memory there, and runs the new version.

BIN — (CSS-B) Copies HZ from Dock to RAM and starts up that version. You can start this version with RAND USR 32776 if you load it from tape. A warm start is still RAND USR 24098.

HOOKUPS

CSS-M
CSS-P — User hook-ups to the HOT Z command interpreter. ENter the address of a routine at 5F90, And the PI key causes a jump to that address Enter the address to 5F92, and the TAB key will cause a jump to that address. Addresses entered must not lie in the range 8000-BFFF. See the introduction for an explanation of how to call that memory range.

WRITE Mode Commands

ESCAPE

SS-O
; key — Escapes without change during assembly edit.

HEXEDIT

SS-E — Switch to hex-edit mode from assembly edit. Moves the cursor horizontally.

ASSEMBLE

SS-A — Switch to assembly-edit mode. Works only when disassembly display and edit mode are on. Moves the cursor horizontally. Doesn't work with the data display because assembly doesn't apply to data.

DELETE — Deletes the instruction at the cursor and closes up the code between the cursor and END. END may be either lower or higher than the cursor address. If END is less than the cursor address, then code is moved from lower addresses to close the space; if END is greater than the cursor address, then code is moved from higher addresses to close the space. Code at the END address and beyond (moving away from the cursor) is preserved. If END is 256 or more bytes away from the cursor, then you will be asked each time to verify the END value before the command is executed. The purpose of this is to prevent your messing up the entire RAM by forgetting to set END properly.

EDIT Sets the Insert mode for the next instruction (only) to be entered. If END is less than the cursor address, then instructions are pushed to lower addresses (up the screen) as far as END; if END is greater than the cursor address, then instructions are moved to higher addresses (down the screen) as far as END. Any NAMEs assigned to shifted memory area will also be shifted so that they stay with the instruction to which they were assigned. Relative jumps to or from the shifted area are not corrected and may require a fix-up. If END is 256 bytes or more from the cursor address, you will be required to confirm the END value before the operation proceeds.

ENTER Quit to READ mode when cursor is in "home" column. During hex entry, ENTER escapes and leaves the original memory contents intact. During mnemonics entry, ENTER sends the line contents to the assembler for entry into memory.

STEP

STEP Single-steps the instruction at the cursor address and switches to the single-step display with the result of of that instruction in the register values and the following instruction in the NEXT slot.

SET END

TO Brings up the END? cursor that allows you to reset the END variable. Whenever a block of code needs to be marked, it is generally delineated by the cursor address and the address assigned to END. Always use it to block out a segment of memory for Insert and Delete commands before beginning to edit. END should be set within 256 bytes of the cursor for editing, but that restriction can be overridden in any particular case. (See Insert and Delete instructions.)

OR Sets END equal to the current cursor address.

FIND STRING

CSS-F FIND the string marked by the cursor (first byte) and END (last byte). Sets the display to start with the found string. If no match is found, then the display remains at the template string. To find the next match without going back to the template, use CSS-G. Do not use other commands between these two.

FIND NEXT OCCURRENCE OF STRING

CSS-G FINDs the next successive match to the template string set up by CSS-F. After a match is found, you must move the cursor past the beginning of the matching sequence before using this command, to avoid finding the same occurrence again.

ASSIGN NAME

CSS-N NAME command. This command has two separate effects, depending upon whether it is used with the disassembly display or the data display. With the disassembly display, the effect is to christen that instruction with the NAME that you enter to the screen following the command. A NAME requires four characters with at least one beyond F in the alphabet. (All of lower case works.) Space and semicolon should not be used. With the data display, the NAME you enter following the command must already be assigned to some address. HOT Z then looks up the address for that NAME and pokes that address to the byte at the cursor address and the byte following, then moves the cursor down two bytes. Use this form for entering tables of addresses

DELETE NAME

CSS-X Deletes the NAME at the cursor address from the current NAME file. This command will only affect the NAME that you see on screen with the disassembly display, so it is best not to use it with the data display. Do not attempt to use this command before you have moved the NAME file to RAM with the NSET command.

CLEAR MEMORY

ERASE Clears memory from cursor address to END. Works only on unprotected RAM.

FILL MEMORY

FN — Fills memory from cursor address to END with the code for a key that you specify in response to the KEY? prompt. For unkeyable values, write that value to the HOT Z variable FILC (5FA4) and then use the ERASE command.

CASSETTE COMMANDS

SAVE CODE

CSS-SAVE — SAVEs code from cursor to END-1. Enter a tape name without quotes. This is a CODE-format SAVE. You can reload such tapes from BASIC by converting the cursor address to decimal and setting the byte length to END minus cursor address. From Home bank only.

VERIFY

VERIFY — VERIFIES a CODE format tape from cursor to END-1. No quotes on tape name. Compares with Home bank.

LOAD CODE

CSS-LOAD — LOAD from cursor to END. Loads 2068 CODE-format tapes. Set the cursor to the start address and END one byte beyond the last, such that END minus cursor address equals the byte length. Unlike the BASIC command and earlier versions of HOT Z, a tape name is always required by this command. No quotes are used. Loads to Home bank.

TRANSFER COMMANDS

CSS-T — Transfers memory content (either within or between banks of memory) between the cursor address and END (inclusive) to a destination (DEST) that you enter following the command. First enter source and destination Banks. (00FF means from Dock to Home.) Hit ENTER for a default to FFFF, which means Home-to-Home. Then put in the Destination address (DEST) in the bank you want the stuff to end up in, if that's not too many 'in's.' The ENTER key after DEST executes the command; SPACE after DEST cancels the command; TO after DEST lets you reset END before the command is executed. Does not transfer NAMEs. To do that, use the MERGE command, which is otherwise identical to this one.

CSS-SS-T MERGE)

TRANSFER memory contents and assigned NAMEs from a memory block (cursor address to END, inclusive) to an area beginning with an address entered in response to the DEST prompt. (See CSS-T command.) This command depends on the NAME file being in Home RAM; do not attempt to use it until you have done an NSET. (Should NSET be part of initialization?) This command is not so often nexessary, except for small block moves.

DIS/DAT

CSS-GOTO

Display switch, data/disassembly. Works only from hex-edit mode. (THEN key) Answers most of your decimal to hex perplexities, reads BASIC and ASCII in rightmost column.

RUN IT

CSS-RUN

Runs code beginning at the cursor address. Returns to HOT Z with the first RET. If you do an extra POP and destroy the return address, then you are on your own. (This command expects to jump to the bank structure described by DISB, Home by default, but whatever you set it. If you set a new bank, then then you must set the return which requires a JP back to HOT Z in Bank 0, chunks 4 and 5.) Recommended procedure is to test your routines first with the single-stepper before attempting the R command.

CHECKSUM

LEN

Performs a 32-bit CHECKSUM from cursor address to END and switches to the STEP display, where the sum is in BCDE.

HEX ARITHMETIC

CSS-A

Does hex arithmetic. Takes the cursor address (K) and END (E) and displays on the top line the sum (E+K) and difference (E-K) in hexadecimal. Bank indifferent.

PART SCREEN

AT

Moves cursor to far left of screen and awaits your entry of an address, then disassembles from that address to bottom of screen. Use it for a composite listing. Use CSS-COPY immediately after to print the screen display. Depends on the Bank-chunk description in DISB for what memory it reads. Therefore, any screen that can be printed will be all in one bank.

CODE RELOCATION COMMANDS

MOVE Relocates Z80 code between the cursor address and END. Readdresses all CALLs or JPs. Allows a three-way partition of code, variables and (constant) files. Requires nine addresses to be first entered at TEM1 through TEM9. TEM variables are in the permanent NAME file and cohabit with inessential BASIC variables. Set them before you use the command. TEM1 through TEM3 are the start address of the code block, the end address of the code block, and the destination address of the code block. Cursor and End are usually set to the first two of these, and the third is the DEST. TEM4, TEM5, and TEM6 are usually the file block of constants associated with the program, and TEM7, TEM8, and TEM9 are generally the block of variables, or reserved temporary memory space, where the only important thing is the address. HOT Z assumes that these three blocks can be moved independently. If there are blocks you don't want to touch, then you can use 0000 as a default value to any block of three TEM values.

CSS-Y READDRESS a jump table (address file) between the cursor address and END by a 16-bit displacement value entered in response to the DISP prompt. Takes the address (lo-hi order) at each pair of memory locations, adds the displacement, and re-enters the sum to the same locations.

CSS-U READDRESS that portion of a NAME file between cursor and END by the value you enter to DISP. For special file manipulations only. Normally, you should use the MERGE command to move NAMEs and code around in memory.

PRINTER COMMANDS

CSS-COPY COPIES screen to 2040 printer. Intended mainly for use with the PARTSCREEN command for printing out composite disassembly from separate address blocks.

LLIST Outputs the screen and beyond without headings from the cursor address to END to the 2040 printer.

EPROM BURNING

FLASH Burns an EPROM on the Oliger EPROM burner. Format follows the Transfer command. Code from cursor to End is burned to the DEST address on the chip (normally 0000, but you burn as little as a single byte). You are prompted for the type of chip (2764 or 128) and for burner Power On and Power Off; flip the switch and hit a key when ready.

CREATING AND PRINTING ASCII FILES

POINT   Creates an ASCII source file that could be edited and used with an assembler. The code in whatever bank is active is disassembled, the address and hexcode columns are discarded, and the lines are terminated with a semicolon. The ASCII codes are sent to a file in HOME bank at the address determined by the pointer DES2 (SFD8). You must set that pointer manually by writing directly to it with HOT Z. At the end of the operation DES2 will point to the end of the file, so you could use this command successively to create one file from several separate segments of memory. The disassembly begins with the cursor address and finishes at END, which must be set in advance. You must have assigned a label to every jump or call address if you expect the file to be palatable to an assembler.

CODE   Creates an ASCII file of everything on the HOT Z screen from the cursor address to END. The file is created at the address contained in DES2 (SFD8) in HOME RAM. At the end of this operation, DES2 will point to the end of ASCII file. Move that address to END with the TO command for printing, saving, or viewing the file. You should be able to get at such files with a word processor in order to add annotations for archiving. Be sure to set DES2 low enough that your file will fit below FFFF, as there is no check for overflow.

SCREEN$ Prints an ASCII file to screen. Set the cursor to the first byte of the file and set END where you want to stop. Printing will pause for the Sinclair "scroll?" after 22 lines, and you can break with the space bar or continue by hitting ENTER. This command is for viewing only; it does not allow you to edit the file.

LPRINT   Prints an ASCII file to a line printer. Prints from the cursor address to END. You can interrupt with the BREAK key.

FILE COMPARE

CIRCLE Set the cursor to the first address of one block and set END to the first address of the block to be compared. (This could be the same address if the blocks are in different banks.) When you give the command, you will be asked for source and destination banks: enter the banks of the two memory blocks. The comparison will begin at once and the display will jump to the first address where the memory contents differ. It important that you only use this command with the cursor set at an EVEN (0,2,4,6,8,A,C,E) numbered address. The purpose is to find small differences in two blocks of code; the command will not be useful for blocks that differ greatly.

HOOKUPS

CSS-M User hook-ups to the HOT Z command interpreter.
CSS-P Enter the address of a routine at SF98, and the PI key causes a jump to that address. Enter the address to SF9A, and the TAB key will cause a jump to that address. Addresses entered must NOT lie in the range 8000-BFFF.

SINGLE-STEP MODE

Key Function

QUIT

SS-Q Quit single-step and return to READ. Return address is the address in the NEXT slot of the single stepper. Register values will be preserved if you reenter from READ mode.

STEP

ENTER Runs the instruction in the NEXT slot and reports the resulting register values.

SPACE Skip the step in the NEXT slot and advance to the next instruction. Skipped instructions are not listed in the LAST slot at the top of the disassembly segment.

EDIT Backs up. On its first use, this command takes the instruction from the LAST slot at the top of the disassembly listing and puts it in the NEXT slot (second line). Repeated use with no intervening commands will back up one more byte for each keypress. Intended use is just to get the last step back.

PRINTOUT

CSS-COPY Print screen. Copies current screen to printer.

RUN IT

CSS-RUN Run a CALL or RST 10. It is your responsibility to know that the called routine will not crash and not to send RST 10 any unprintable characters. The purpose of this command is to shorten the time needed to step through complex routines.

SET BREAKPOINTS

OR Set Breakpoint1. Breakpoints are set just as register pairs are, with a NAME or address entry into the NEXT cursor. You must set the breakpoints precisely to the beginning of the instruction at which you want the single-step to stop, because the stop depends on the address of the next step being exactly equal to the breakpoint. If the breakpoint points to the second byte of a two-or-three-byte instruction, you routine will never stop until you crash or hit BREAK.

AT Set Breakpoint2. Breakpoints are set just as register pairs are, with a NAME or address entry into the NEXT cursor. You must set the breakpoints precisely to the beginning of the instruction at which you want the single-step to stop, because the stop depends on the address of the next step being exactly equal to the breakpoint. If the breakpoint points to the second byte of a two-or-three-byte instruction, you routine will never stop until you crash or hit BREAK.

AND Display the two breakpoints on the line below the flags display.

SS-GOTO Go (run) to breakpoint. Causes the test routine to run from the address in the NEXT slot to either of the two breakpoints, which must be set in advance of this command. Breakpoints must be set to an address that starts a command and not to a byte embedded in a command. The GO routine checks the BREAK key after executing each line of code, so you can recover from endless loops and sometimes from runaway routines (if you're quick) by hitting BREAK.

REGISTER SET

VAL Set register value. The response to this command will be REG? in the NEXT cursor. You should respond as follows for the various registers:

A for the A register
B for the BC pair
D for the DE pair
F for the Flags register
H for the HL pair
S for the user's Stack Pointer
X for the IX pointer
Y for the IY pointer

Note that all settings are 16 bits (two bytes) except for the one hex byte for A and the mnemonic setting for F. The specific flag bits are set or reset with the same mnemonics as are reported (M, P, Z, NZ, PO, PE, C, NC). Use this command to set up initial conditions for testing your routines. Note that you can set the user's SP this way.

ASSEMBLY

SS-A Sets the assembly cursor at the instruction in the NEXT slot so that you can EDIT it. Return to STEP operation with ENTER.

SPECIAL DISPLAY SCREEN

ATTR SETs a second display file (WINDOW) starting at the address in NEXT and extending 1B00 bytes. Any stepped display instructions then output to the window, which comes up before the next register display. Be careful not to erase valuable code by setting the window on top of it. Dismiss the screen with any key but V.

SCRS Toggles the feature that causes the WINDOW to wait for a keystroke before going to register display.

OUT Switches the window out of the STEP loop so that subsequent steps have no effect on it.

IN Switches a WINDOW from OUT back IN again. WINDOW must be SET up first.

HOOKUPS

CSS-M User hook-ups to the hot z command interpreter.

CSS-P Enter the address of a routine at 5F94, and the PI key cause a jump to that address. Enter the address to 5F96, and the TAB key will cause a jump to that address. Addresses entered must not lie in the range 8000-BFFF. See the introduction for an explanation of how to call that memory range.

SINGLE-STEP WINDOW COMMANDS

The single-step window is a utility designed for developing display code. Its use is very tricky and requires that you first acquire some general competance in using the single stepper. It enables you to create, save, and see a special screen, but very painstakingly.

There are four commands, and they are all called from the single-step display (unlike HOT Z-II). You must first have 1B00 (6912 decimal) bytes available for the extra screen.

The commands are:	Key
WINDOW SETUP	ATTR
WINDOW IN	IN
WINDOW OUT	OUT
WINDOW STOP	SCREEN$ (toggle)

All of these are commands whose work goes on behind the scenes. The acknowledgement that the command has been executed is the same in each case, the appearance of a W near the left end of the LAST-NEXT line above the code section of the single-step screen.

WINDOW SETUP establishes an initial white screen and will destroy anything you have in the selected 1B00 bytes of memory. Set up the beginning byte by entering its address, so that it comes up in the bright line of the single stepper. Then give the ATTR command. The SETUP switches the window IN and sets the STOP as well. The initial print position is the top left corner, but don't forget to initialize that in your program for the day you expect it to run by itself.

WINDOW OUT switches the window out of the single step loop but does not destroy it. Any code steps you execute after WINDOW OUT will have no effect on the second screen. The point is to stop it flashing on every time.

WINDOW IN countermands OUT and brings back a previously established window. It will not function if you have not previously set up a window. However, if you have previously been using a window and have reclaimed the space for something else, and if you then use the IN command, you may get some strange effects. If there has never been a window, you will not get the "W" response.

WINDOW STOP is a toggle switch. Each time you press it, HOT Z responds with a "W" on the LAST-NEXT line. When you initialize a window, the stop is set so that the new screen comes up and waits for a keystroke before returning to the register display. If you toggle the stop, the second screen will flash on and then get put away without waiting for a keystroke. Toggle again and the stop will be reinstalled. The point is to switch out the stop for steps that don't affect the display.

There is one subcommand available during the STOP. If you press the V key (CLS), the screen will be cleared and you will be reinitialized to a blank screen and your print position reset to top left.

The WINDOW routines respond only to the print position in S_POSN, not to DF_CC. The latter is always set from the former via a CALL 0914, on every step. If the window is IN when you change S_POSN, then the new screen position will be remembered next time an actual print occurs. In fact, you should always use a window when you do things with S_POSN, so that your manipulations don't mess up the single-step screen.

If you print with RST 10, then you should use the INT (RUN CALL) command to get all the way through the RST in one step. In general, the most effective use of the window will occur when you set up your display routines as subroutines and run through them in a single step with the INT command. Alternatively, you can set breakpoints and use the Run-To-Breakpoint (THEN) command to get through your screen manipulations in one quick dash.

Note that you can save any screens you are working with by exiting the single step and using the HOT Z data save. You will not get a SCREEN type tape from it. (You could set up a block move to screen memory and call that from BASIC along with an in-program SAVE SCREEN$.) Then for re-use, first set up a new window screen from the single stepper, then exit and load in the data tape to the window screen address.

ON NAMES AND NAMING

HOT Z's labelling or naming system is intended to make the programs you are reading or writing more comprehensible when they are listed. The four-letter limit is imposed by the 32-column display. A space is not a legel character in a HOT Z NAME, so use a dash or other punctuation if you want fewer than four letters. A semicolon is also illegal, since it is the escape character for the assembly editor.

The NAMEs themselves and the addresses they assigned to are contained in a special file, referred to as the NAME file. A NAME file is an ordered list beginning with the highest address to which a NAME is assigned (two bytes), then the four letters of that NAME, then the next highest address, etc. After the last NAME in a file, there must be two zero bytes. HOT Z takes care of ordering the NAMEs for you.

HOT Z includes a NAME file that annotates the entire HOME ROM, the system variables, and HOT Z's variables. You will find a few extras among the system variables. TEM1 through TEM9 are slots for temporary 16-bit variables for various HOT Z routines. (You may use them for any of your own routines for values that are not required once the routine is over, provided your routine does not call the floating-point calculator.)

The permanent NAME file that loads with HOT Z can be expanded to hold any NAMEs you add in a session of using HOT Z, or you have the option of starting a new file from scratch. There is room for 192 NAMEs in the existing file. The NAME file must be in RAM before you can add to it. If you are running in Home RAM or the Dock bank of an Aerco board, you can just add or delete NAMEs. If you use a NV RAM board, you must either unprotect it or proceed as if you are using an EPROM cartridge.

If the NAME file is in EPROM or protected RAM, you must open an empty file in RAM with the NSET command (INKEYS in READ) before you try to add NAMEs. The file is opened at the top of RAM. Use the RND command in READ to find the start of the file. After you move it to RAM, you can put it anywhere above E000 or below 8000. The variable ALNA is listed to assist switching file locations. You might also want to copy some of the NAMEs from the permanent file to the RAM file. Use the ordinary transfer command (RND) and transfer from 00 to FF. Easiest is to set the END address to the lowest byte you want (the first address byte of any NAME) and the cursor to the highest (the fourth character of any NAME) and then use FFFD for DEST. The display after transfer will then show the first address of your copied file. Put that at ALNA (lo-hi) followed by FE followed by FF. Then go back to READ and give the OVER command to turn on the new list.

If you try to erase a NAME while the file is in EPROM, you will confuse the look-up and lose the use of the entire file until you reinitialize.

The labelling system has not been partitioned to be multi-bank. A NAME shows up at its address no matter what bank you are in. with a little experience, you will learn to switch between alternate files, which overcomes this problem.

Add a NAME to the file with the CSS-N command in WRITE mode with a disassembly (not data) on screen. The command will give you a cursor in the NAME column and allow you to enter or replace the NAME for that address. A legal NAME is made up of any four single characters with the restriction that at least one character must be beyond F in the alphabet. If you forget that rule, HOT Z will refuse to accept your new NAME and will ask you for another. A space in a NAME will be accepted and the disassembler will list the NAME, but you will not be able to use such NAMEs when working with the assembler. which parses according to spaces and punctuation. Take care that your NAMEs are unique, or HOT Z will always find only the one at the higher address when you refer to it. (If you enter a NAME to the ADDR cursor before you assign it, then the NAME file will be searched and the display will move to that NAME if it is already there; otherwise the display stays put.)

The CSS-X key (WRITE) will delete a NAME at the cursor address from the screen and from the NAME file.

The CSS-T command (READ) is there to let you find the start of your current NAME file. You may want to check up on it if your are working under crowded memory conditions to be sure the file doesn't overwrite some valuable code. This command switches the display to data and moves to the lowest address of the NAME file. Since the NAME column in the data display lists NAMEs assigned to addresses formed by pairs of bytes in the hex column, the NAME appears horizontally across from the first address byte and then vertically opposite the last four data bytes. (Be aware that chance occurrences of data can look like addresses and cause spurious listings in the NAME column of the data display.)

You should also use the CSS-T command when it comes time to SAVE the NAMEs you have entered in a session. However, you will also need to know the end address of your file in order to SAVE it. You can call up that end address by entering NEND to the ADDR cursor; the end address of the NAME file is listed lo-hi there. You can either add 2 to that address to include the two zero bytes that act as a terminator, or you can remember to zero those two bytes after you reload the tape. If you choose the first option, hit RND, turn on the edit cursor, set END to NEND+2, and SAVE. Record the addresses for use when you reload.

When you reload a NAME file, you must install the start and end addresses so that HOT Z will know where to look for that file. This is done at the four-byte block labelled ALNA (alternate NAMEs). With the data display and the edit mode, write the start address (lo-hi) followed by the NEND address; don't forget to subtract 2 if you have included the terminating zeroes. (If you have not included them, make sure they are there first.) If you don't do these settings correctly, you will hang up the program when you try to switch the new file on.

The NAME-file switch command is OVER in READ. It will switch from the permanent NAME file to the one you have loaded, after you have installed the file parameters at ALNA. If you use OVER without installing the new parameters, the effect will be to switch off the NAMEs entirely and you will not be able to add new ones. You should switch off the permanent NAME file in this way before loading a new file; then install the start and end addresses of the new file at ALNA and use OVER to switch them in.

You can amalgamate NAME files only if they pertain to separate blocks of memory, with the addresses in one block all higher than those in the other. Then just load the two files end to end in the proper order and save them as a single file.

To start a completely new file, put the starting/ending address (the same, because it's empty) in the four bytes at ALNA and give the OVER command, then enter NAMEs.

You can SAVE a NAME file as data, then LOAD it in and hook it up by writing the starting and ending address at ALNA and using OVER. Always remember that there must be two zero bytes above the value you assign to the high end of the file.

USING THE RELOCATE COMMANDS (MOVE, STR$, CHR$)

The Relocate command is rather complex in order to provide you a degree of flexibility in relocating your routines. A set of nine addresses must be entered before using the MOVE command, and a certain amount of planning and knowledge of the subject program is required to derive the correct addresses. Simple programs with one or two calls or absolute jumps are best labelled, moved with the Transfer-with-NAMEs (MERGE) command, and then fixed up by hand.

A program of reasonable complexity will have a block of code, a block of data (which may include address lists or jump tables), and a block of variables. Good programming form would recommend that you keep these blocks separate and distinct rather than, say, mingle data and variable storage in the crannies between your subroutines. If you are programming with HOT Z, you can separate the blocks generously as you develop your program and then use the Relocate command to close the gaps when you finish.

HOT Z's Relocate command will work on program blocks where code, data and variables are separate and distinct. If you have embedded patches of data, the command may still work, but you should check the data after the relocation to make sure that it has not been changed under the guise of readdressing code. Programs such as the 2068 ROM, where jump tables lie around like empty beer cans, would have to be broken up into segments and relocated piecemeal.

The Relocate routine readdresses and moves Z80 code. However, the command does not take account of overlapping segments between source and destination blocks, so you cannot directly relocate a program to addresses already occupied by that program. (In such cases, you should use the transfer command first and then readdress in place with the relocate command.)

Jump tables have to be revised with the CSS-Y command, which first asks you for a displacement and then adds that displacement to each address in the file, starting at the cursor and ending at the END address. (If you moved your code from 8100H to 8400H then the displacement would be 0300H; from 8400H to 8100H would be a displacement of FD00H.) Jump tables and data blocks should be moved with the Transfer command prior to using the relocate command.

The Relocate command (MOVE) allows you to move the code block by one displacement, the data block by another, and the variables block by a third displacement. (Any other three-way separation should also work.)

ADDRESS ENTRY FOR RELOCATING

The variables TEM1 through TEM9 are used to set the nine address parameters for relocation. The nine addresses are three sets of three addresses. Each set of three addresses indicates the start and end of an address range to be changed and the start address of the new address range. For example, suppose your program to be relocated fits the following memory map:

84D0-84E8	Variables
84F0-84FF	Data
8500-8680	Program

Suppose you want to put the variables and data at 8100H and the program at AC40. First, transfer the variables block to 8100H; it will run to 8118, so transfer the data block to 8119-8128. To move the program from 8500 up to AC40, any addresses of jumps or calls that lie between 8500 and 8680 should be changed to lie between AC40 and ADC0. (You don't need that last number.) So enter the original range in TEM1 and TEM2 and the first address of the new block in TEM3, thus:

TEM1	8500
TEM2	A680
TEM3	AC40

These first three TEM values always hold the parameters relating to the program (code) block. Variables and data parameters can go interchangeably into TEM4-TEM6 or TEM7-TEM9.

Addresses of variables, which were at 84D0-84E8, must be changed to start at 8100, and addresses of data, formerly at 84F0-84FF, must be changed to begin at 8119, so fill in the remaining TEM slots as follows:

Variables		Data	
TEM4	84D0	TEM7	84F0
TEM5	84E8	TEM8	84FF
TEM6	8100	TEM9	8119

TEM4-6 are one block, TEM7-9 the other. Now set the cursor at 8500 (start of the code segment) and set END to 8680, then give the MOVE command. The code will be copied to the new location and readdressed to run with the new variables, new data block, and any relocated subroutines in the code block. The original code will remain unchanged at its original location.

You may also use the Relocate command to split a code block into two or more separate blocks, but you must apply it repeatedly, once for each of the end-product blocks, and readdress for the blocks that are not being moved as if those blocks were variables or data.

If you lack variables or data blocks, then use a single non-zero dummy value for all three of the second or third set of TEM values, i.e., make them all three the same.

The relocator leaves unchanged any ROM calls or any loads to or from the systems variables area (5C00-6000).

After you have relocated a program, you may want to readdress a block of NAMEs that pertain to it. The command on the CHR$ key will do this for you. The CHR$ command works just like the STR$ command, except that it readdresses every third pair of bytes. Just enter the proper displacement. If you are readdressing only part of a label file, you may have to do some block moves to keep all the addresses in inverse sequence. Labels will be lost (from the screen, not the file) if you destroy the ordering of the addresses.

Appendix A

THE FLOATING-POINT INTERPRETER

RST 28H is the entry into the ROM's floating-point operations, which are coded in the bytes between an RST 28 and the following 38H. There is a good explanation of this second language (Or is it third?) of the ZX in Dr Logan's article in SYNC 2,2. (But beware of the two sign tests, which aren't jumps, as labelled in SYNC.) Note also that there have been a few changes for the 2068 ROM.

HOT Z will read this floating-point language, but only after you turn on the floating-point interpreter (CSS-O in READ). If you leave the floating-point interpreter turned on, you will get a true reading of the ROM, but problems can arise elsewhere in memory when you encounter an EF that functions as data rather than an RST 28. You may get locked into the floating-point interpreter mode, without a 38H, the END character, in sight. The way out from this barrage of gibberish is the CSS-O command again, which switches out the floating-point interpreter entirely. Other times you may want to read it, because this extra language is really one of the treats of the Sinclair-calculator heritage.

The f-p interpreter is also turned off by entry of a numerical address, but not by a page flip or a NAME, so use the last two when you're working with f-p. In addition, there is a special key command, CSS-I in READ mode, which switches the flag that tells the disassembler which language it's in.

The CSS-I command (READ) has a dual purpose. It will get you out of floating-point mode (without turning off the interpreter) if you need to and can't, or it will get you in when you want to be but aren't. You may get stuck in that mode through addressing yourself into the middle of a Z80 instruction, for example. Since floating-point operations include jumps and loops, there are also inclusions of f-p code that do not begin with an RST 28, branches of jumps. The CSS-I command will get you into those branches. However, the command is just a bit switch and it doesn't function when the screen page itself switches from one language at the top to the other at the bottom. The cure, when the CSS-I command doesn't function is the trick of hitting the THEN key twice. This picks up the language mode from the bottom of the page to the top and reverses the reading of any bytes from one language to the other.

You will also encounter some queer behavior if there is f-p code at the bottom of the screen and you try to write or go to the One-Step. This is not generally fatal and can be cured by going back to disassembly and setting the screen so that it ends in Z80 disassembly. If you want to write f-p code, the only manageable way is to go into EDIT mode in data.

The two data-stacking operations are labelled STFP (stack floating point) and APPX (approximator) The first of these puts one five-byte number on the calculator stack, the second a series of one to 31 (whatever is left when you AND the low nibble of the instruction byte with CF) five-byte f-p constants. (That's 5 to 155 bytes.) The approximator uses anything from six to a dozen floating point constants to get to a value for Chebyshev polynomials to approximate the transcendental BASIC functions.

Floating-point operations are FORTH-like stack manipulations and easy to follow if you know something about that language. They use the MEM area of the systems variables as storage slots for six floating-point numbers. (Each is five bytes.) The f-p operations that transfer between the calculator stack and MEM are called GET and STOR and are followed by a single digit from 0 to 5 to indicate the slot used. Numbers or letters higher than 5 generally indicate a patch of nonsense with GET, STOR and STAK as well.

Many of the possible f-p operators do not occur in the coding of the ROM, where you are likely to encounter them with HOT Z. They occur instead during the ROM's reading of BASIC programs, and they are generally identical with a BASIC instruction. You could learn to write floating-point code with these and the purely machine-code f-p operators if you wanted to; it would be similar to BASIC and a little faster. The 'entry point' of these BASIC f-p operators into the real machine world is through the operation labelled RAFP (Run A as Floating-Point). However, you need only use the command numbers listed as the first column of the instruction list to perform those BASIC functions on whatever floating-point numbers are on the calculator stack. From the perspective of a HOT Z user, RAFP would be used only to run an operation that resulted from some calculation, whose result was a code in A.

Two of the f-p operations deliver data directly from the code listing to the calculator stack. They generally do this in an efficient way, using fewer than five bytes, if possible, to encode the five-byte floating-point number. HOT Z prints the encoded floating-point number in the NAME and mnemonics columns of the disassembly listing. Since the interpreter doesn't know where any number will end, it is necessary to begin all of them slightly out of column, or the longest would run into the next line and mess up the display file. The f-p interpreter also reads the full five hex bytes that go onto the f-p stack, rather than the condensed version that actually occurs in the ROM. The ADDR column keeps accurate track, and you can work out the extra bytes, which are generally trailing zeroes, from that column.

HOT Z prints floating-point data by using the same ROM routines that handle that data, so the disassembly slows down and becomes jerky when it has to print those huge numbers, or their single-digit versions.

FLOATING POINT OPERATIONS

Code	Op	Addr	Description
ØØ	JRT	3AAA	Jumps if stack top holds a true
Ø1	SWOP	37FB	Exchanges the top and second 5-byte stack entry
Ø2	DROP	376Ø	Throws away top stack entry
Ø3	SUB	33CE	Subtracts top stack from second stack entry
Ø4	MULT	3489	Multiplies top two stack entries and leaves product on stack
Ø5	DIV	356E	Divides second entry by top stack, leaves quotient on stack
Ø6	PWR	3C6C	Raises 2nd on stack to power of stack top
Ø7	OR	3936	Performs BASIC OR on two top stack entries and leaves result
Ø8	AND	393F	Performs BASIC AND on two top stack entries, leaves result
Ø9	N<=M	3956	Numeric inequality test
ØA	N>=M	3956	Numeric inequality test
ØB	N<>M	3956	Numeric inequality test
ØC	N>M	3956	Numeric inequality test
ØD	N<M	3956	Numeric inequality test
ØE	N=M	3956	Numeric equality test
ØF	ADD	33D3	Adds two top stack entries and leaves sum on stack
1Ø	$AND	3948	ANDs a string with a number
11	$<=	3956	String inequality test
12	$>=	3956	String inequality test
13	$<>	3956	String inequality test
14	$>	3956	String inequality test
15	$<	3956	String inequality test
16	$=	3956	String equality test
17	$TR+	39B7	Concatenates strings addressed by the two top stack entries
18	VAL$	39F9	BASIC Function
19	USR$	38D7	BASIC Function
1A	RDIN	3A6Ø	Read in data from channel in A
1B	NEG	382D	Changes the sign of top stack entry
1C	CODE	3A84	Replaces top stack entry with its sinclair code
1D	VAL	39F9	BASIC function
1E	LEN	3A8F	BASIC function
1F	SIN	3BDØ	BASIC function
2Ø	COS	3BC5	BASIC function
21	TAN	3BF5	BASIC function
22	ASN	3C4E	BASIC function
23	ACS	3C5E	BASIC function
24	ATN	3BFD	BASIC function
25	LN	3B2E	BASIC function
26	EXP	3ADF	BASIC function
27	INT	3ACA	BASIC function
28	SQRT	3C65	BASIC function
29	SGNM	3851	BASIC function
2A	ABS	3829	BASIC function
2B	PEEK	386B	BASIC function
2C	INX_	3864	BASIC function

2D	USR#	3872	BASIC function
2E	STR$	3A3A	BASIC function
2F	CHR$	39E4	BASIC function
3Ø	NOT	391C	BASIC function
31	DUP	377F	Duplicates top of stack (5 bytes)
32	QREM	3ABB	Replaces number pair by quotient on stack top, remainder below
33	JRU	3AA1	Unconditional relative jump
34	STFP	3785	Composes and stacks number from following data bytes
35	LONZ	3A95	Loop jump as DJNZ with BERG as counter
36	N<ØØ	3921	Tests sign of stack top and replaces with true if negative
37	N>ØØ	3914	Tests sign of stack top and replaces with true if positive
38	END	3AB6	Ends an RST 28 routine
39	AADJ	3B9E	Adjusts angle values modulo 2 pi for trig functions
3A	ROUN	35D3	Rounds down to integer
3B	RAFP	3761	Runs byte in A as f-p op code for BASIC functions
3C	DEXP	31ØD	Decimal exponent processor
8Ø	APPX	38Ø8	Successive approximator; stacks and processes constants
AØ	STAK	37DA	Stacks Ø,1,Ø.5,PI/2,or 1Ø, depending on second nibble
CØ	STOR	37EC	Stores entry in calculator MEM slot given by 2nd nibble
EØ	GET	37CE	Recalls stored entry from calculator MEM slot in 2nd nibble

Hot Z 2068

TS 2Ø68 ROM NAMES

#stm	22ØF	B	Routine to change active stream
$and	3948	FP	Executes AND between string (params on calc stack) and no. on calc stack
$stk	2E6F	B	Stacks parameters for a sliced or array-element string
$tov	2F84	B	Transfers a newly declared sting to variables area
$tr+	39B7	FP	Executes string concatenation for two string params on calc stack
1int	1F1E	B	Gets 1-byte integer from calc stack to A
1num	1BE5	B	Class 6: GOTO,IF,GOSUB,PAUSE,BORDER,OPEN,CLOSE
1num	1BE5	B	Evaluate one expression for command class 6
1spa	12B8	E	Opens one space at area designated by HL
2int	1F23	B	Gets 2-byte integer from calc stack to BC
2num	1BDD	B	Class 8: POKE, BEEP, OUT
2num	1BDD	B	Evaluates two expressions for Class 8 commands
Blis	14E1	B	List the BASIC program to screen
Stop	1C59	B	Error 9 trap for STOP command
aadj	3B9E	FP	Reduces angle size for trig calcualtions; FP op 39
abak	33C3	FP	Adds back the carry when a number is shifted right
abs_	3829	FP	FP op to make last calc stack value positive
acs_	3C5E	FP	Replaces X on calc stack with ACS X
adch	ØAE7	E	Adds a character to EDIT or INPUT line
add_	33D3	FP	Floating point addition of two numbers
adnx	172Ø	B	Finds address of next program line or next variable
alnm	3Ø46	B	Returns C flag set if A hold digit or letter
alog	317F	FP	Gets log base 1Ø of 2 to power A into A
alph	3Ø4B	B	Returns C flag set if A holds a letter
and_	393F	FP	Executes AND on last two calc stack values
arin	17B5	O	Bankswitches for cartridge software (BASIC)
arln	17CF	O	Searches for line no. BC in cartridge
aros	18C6	O	Sets up buffer for cartridge software
asfi	3DØØ	F	ASCII character file (to end of ROM)
asn_	3C4E	FP	Replaces X on calc stack with ASN X
atn_	3BFD	FP	Replaces X on calc stack with ATN X
badr	37C5	FP	Finds base address for each fp form in calc MEM area
basl	1158	E	Adds a new BASIC line to existing program
bcfi	1945	B	BASIC command routine offset table
beep	Ø436	S	Beeps in pitch and duration from calc stack (2 nos.)
blin	15A1	B	Prints a BASIC line for the LIST command
bper	Ø3F3	S	Beeps notes according to values in DE & HL. Callable.
brck	29A6	B	Gets closing bracket and loop to expression scan
brdr	243E	B	BORDER command routine; gets color from calc stack, sets INK
brds	2441	B	Call-in point to set border with color in A (used by HZ)
brek	2ØØ9	O	Reads BREAK key; returns NC if SHIFT-BREAK is pressed
brfl	241D	B	Handles BRIGHT and FLASH (C set for FLASH)
casr	2548	B	Does bank switch to EXROM for cassette routines
cass	24D2	B	Handles cassette commands for cassette or disklike devices
cat_	25C8	B	Supplies CAT token in B
cbuf	ØA23	P	Sends contents of printer buffer to printer
ccfi	Ø528	P	Table of offsets for control-character subroutines
cdpm	27D6	B	Subroutine to set initial parameters for CIRCLE and DRAW

celi	1363	E	Clears edit line
chex	1265	O	Channel exchange routine
chfi	1293	O	Channel-code offset table
chfl	124D	O	Set flags for channel
cho2	123F	O	Find address of channel for given stream
chop	123Ø	O	Channel-open routine (FD-Ø3 as stream no. in A)
chr$	39E4	FP	Replaces X on calc stack by params of CHR$ (X)
cins	12BB	E	Opens BC spaces at address HL
circ	2679	B	CIRCLE command routine
cknd	1B44	B	Syntax check routine; faults to error unless at line end
clØØ	1B73	B	Class Ø; STOP, RETURN, NEW, CONT, CLS, COPY
clØ1	1B82	B	Class 1: LET
clØ2	1BB1	B	Assigns value to variable in LET statement
clØ3	1B7Ø	B	Class 3: RUN, RAND, CLEAR, RESTORE
clØ4	1BCF	B	Class 4: FOR, NEXT command routines
clØ5	1B74	B	Class 5: DEF FN,DELETE,ON ERR,RESET,SOUND
clØ5	1B74	B	Class 5: PRINT,INPUT,DIM,REM,LIST,READ,DATA,LPRINT,LLIST
clØ9	1C29	B	Class 9: PLOT,DRAW,CIRCLE; sets default conditions
clØb	1C46	B	Class ØB: cassette routines
clds	Ø8EA	F	Subroutine to clear display
cler	1F36	B	Executes the CLEAR routine
clfi	1B64	B	Command class routine offset table
clli	Ø97F	F	Clears lower B lines of the display
clno	16E8	B	Compares line no. in BC with (HL), returns Z for match
clo2	13BE	B	Closes channel with channel address BC
clo3	13D8	B	Closes intelligent device
clos	139F	B	Executes CLOSE #N (closes channel)
clow	Ø8A9	F	Clears lower screen (command lines)
clpb	ØA35	F	Clears the printer buffer
clrn	1F39	B	Entry point to CLEAR used by RUN
cls_	Ø8A6	F	Executes BASIC CLS; callable
clsm	14ØD	B	Fetches channel pointer for close-stream routine
clws	ØBFD	E	Clears the editing workspace
cnfi	11AA	F	Initial channel address file
cocl	2416	B	Changes a color system variable according to mask in B
code	3A84	FP	Replaces params of A$ on calc stack by CODE A$
col1	238B	B	Gets next character to sort for color controls
col2	238C	B	Sorts for color item followed by semicolon or comma
col3	239C	B	Subroutine to sort for INK,PAPER,FLASH,BRIGHT,INVERSE,OVER
col4	23A6	B	Reduces color token to control character and sends to screen
colv	23BB	B	Sets color system variables for PRINT
comr	1B79	B	Gets command routine address from syntax table and jumps
cons	3684	F	File of constants in FP form: Ø,1,.5,pi/2,1Ø
cont	1EE4	B	CONTINUE: loads up line and statement no. for jump
copy	ØAØ2	F	BASIC COPY command (callable)
cos_	3BC5	FP	Replaces X on calc stack with COS X
cpfn	2BD2	B	Compares found DEF FN with FN under evaluation
cpit	2E8A	B	Evaluates next expression, compares with limit in HL, gives A=FF if over, else Ø
cpli	ØA4A	F	Copies one pixel line to printer
cret	Ø566	F	Carriage-return routine
crst	2454	B	Checks for cold start symbol after RESET

csfi	14Ø7	B	Table of offsets for close stream routines
ctch	21ED	B	Handles position control characters in PRINT: simicolon, comma, apostrophe
ctem	Ø888	P	Sets temporary color values
cusr	388E	FP	Checks for cartridge and if so sets up banks for USR call
darc	2792	B	Arc-drawing subroutine
data	1E82	B	DATA statement; syntax gets checked, but as REM in run
de+1	2EAC	B	Loads (DE+1) to DE, points HL to DE+2
deck	Ø371	O	Decodes key value according to mode and shift state
defp	3Ø59	B	Handles BIN and converts decimal nos. to fp form on calc stack
dele	28ED	B	Handles DELETE key
delk	Ø9E7	?	Delays and waits for a keystroke (use unknown)
dell	2ØD1	B	Executes DELETE (lines) command
dexp	31ØD	FP	Moves a general E-format decimal to calc stack
dffn	2Ø1D	B	DEF FN command; check for syntax, skipped in RUN
diff	1745	B	Sets BC = HL - DE; returns HL & DE exchanged
dim_	2FCØ	B	Sets up space for new arrays in VARS, reclaims old ones if any
div_	356E	FP	FP division; exits via the mult routine
draw	26DB	B	DRAW command routine (26FC resumes floating point ops)
drop	376Ø	FP	Executes a return to drop a number from the calc stack
dup_	377F	FP	Duplicates a number on calc stack or moves a number to calc stack
echp	ØC83	E	Echoes keyboard buffer to current channel (lower screen)
eddl	Ø87B	E	Handles DELETE during EDIT
eddn	ØB59	E	Handles cursor-down during edit
eder	ØBE5	E	Handles errors during EDIT
edfi	ØBØ6	E	Offset table for edit-key subroutines
edgr	ØBDC	E	Handles graphics codes during EDIT
edit	ØBØF	E	Handles EDIT key functions, including INPUT
edky	ØAF8	E	Handles edit keys during line entry
edlf	ØB6D	E	Handles cursor-left during EDIT
edlm	ØB97	E	Moves cursor toward start of edit-line
edot	ØB64	E	Reads & ignores 2 characters and ends edit in error
edrt	ØB72	E	Handles cursor-right during EDIT
edst	ØB67	E	Handles STOP key during INPUT
edtr	ØA82	E	Editor for BASIC line entry or INPUT
edup	ØBBF	E	Handles cursor-up during edit
efor	3ØA9	B	Converts E-format entries to floating point on calc stack
elno	1768	B	Gets line number of line in edit area to BC
end_	3AB6	FP	End an RST 28 calc and return to Z8Ø language
endp	21E4	B	End of print; tests for), carriage ret, and colon
endv	2F68	B	Adds a character to the end of VARS area and writes a new end byte (8Ø)
eras	25D4	B	Supplies ERASE token in B
ertr	ØØ53	O	Fetches error no. to ERR_NR & resets stack
exp_	3ADF	FP	Replaces X on calc stack by EXP X
fadd	335A	FP	Prepares fp form for addition; complements negatives & replaces sign bit
fcon	37B6	FP	Finds needed constant in table of FP constants via A
fdev	1374	O	Searches config table for device spec in C
fet2	3379	FP	Fetches 2 fp forms; first to H'B'C'CB, second to L'D'E'DE
fiat	28D7	B	Finds attribute at screen coords from calc stack, stacks attribute
fist	16FØ	B	Finds statement D in a BASIC line (or token E)
fito	1D28	B	Finds match for token in E starting at (HL)
flas	16ØD	B	Prints flashing cursors
fmul	347F	FP	Prepares fp form for mult or div; tests for Ø, replaces sign bit

fnev	2BEF	B	Evaluates arguments of an FN using found DEF FN during scan
fnum	1C49	B	Fetches number if there else puts zero on stack
fnva	2C4B	B	Evaluates FN from argument values determined with DEF FN
fnwl	1AEC	B	Finds new line address after a program jump
for_	1C78	B	Executes FOR command with value and limit on calc stack
form	25CC	B	Supplies FORMAT token in B
fpbc	3160	FP	Compresses value on calc stack into BC, C set if too big, Z set if positive
fpen	372B	FP	Re-entry point for the fpop routine
fpfi	3696	F	File of addresses for FP ops. Use data display
fpop	371A	FP	Executes FP ops that follow RST 28. FP op interpreter
fppr	31A1	FP	Prints last value on calc stack to current print position
fpta	3193	FP	Gets number from calc stack to A; C set if overflow, Z set if positive
freØ	2969	B	Jumps to main routine for FREE
free	2934	B	Executes FREE statement
g$st	1BEF	B	Evaluate string expression for command class ØA
g$tr	1BEF	B	Class ØA: FORMAT,MOVE,ERASE,CAT
gars	38CB	FP	Way out of cusr when cartridge is present (for USR)
gatr	Ø9C3	P	Returns attribute address (DE) for given display addr (HL)
get2	1FØF	B	Gets two values from calc stack to A and BC
get_	37CE	FP	Gets fp no. from calc MEM area to calc stac (getØ to get5)
gint	313D	B	Gets a small integer (- to +65535) from (HL) into DE; sign in C reg
gkey	11CF	E	Gets keyboard input during INPUT and EDIT
gosb	1F99	B	Executes the GOSUB command
goto	1EF1	B	GOTO: gets and tests line number for jump
gstk	2FAF	B	Reads out the calc stack into BCDEA
gtp2	Ø634	P	Get current printer position parameters
gtpi	29E5	B	Puts PI on calc stack
gtpo	Ø61A	P	Get current print position parameters
gva2	1BBC	B	Evaluates expression to get value for INPUT
gval	1BB9	B	Evaluates expression to get value for LET or READ
hl*d	2EB2	B	Sets HL= HL*DE; gives error 4 if overflow
if__	1C5B	B	Executes IF command on last calc stack value
inas	2363	B	Subroutine to assign an INPUT value to a variable
indx	136B	O	Indexes into tables for various look-ups
infp	3ØF9	B	Puts line no. or integer in BASIC line on calc stack
init	ØD31	O	Main initialization routine when 2Ø68 is switched on
ink$	29F2	B	Executes INKEY$; stacks input string or empty string
inpa	11E1	E	Saves registers and points HL to input address
inpl	2282	B	Handles INPUT LINE
inpr	226B	B	Handles control items during INPUT
inps	2297	B	Handles simple input variables
inpt	222B	B	Main input routine; opens channel K
inst	237A	B	Handles STOP in an INPUT line
int_	3ACA	FP	Replaces X on calc stack by INT X; 3ad2 continues FP code
inx_	3864	FP	Puts result of IN X onto calc stack
iprm	22A4	B	Put INPUT prompt into workspace, gets input and assigns it
jrt_	3AAA	FP	Jump relative on true on calc stack; FP op ØØH
jru_	3AA1	FP	Jump relative unconditionally; followed by offset; FP op 33
kbsc	Ø2BØ	O	Keyboard scan, returns Ø-39d in E, shift state in D
kcha	129A	O	K channel (lower screen) flag set routine
kend	Ø32E	O	End of keyn routine if a key pressed
keyn	Ø2E1	O	Main keyboard read and decode; key to LAST_K, set 5, FLAGS

klft	Ø53A	F	Cursor-left routine
krep	Ø336	O	Repeating key routine; sorts for tokens and DELETE
krgt	Ø554	F	Cursor-right routine
ksca	ØC15	E	Scans keyboard and returns keycode (Try it)
kyfi	Ø227	F	key tables for interpretation of keyboard modes.
ladr	Ø9D6	F	Gives display address (HL) for screen line (B)
ldr2	2813	B	Use as entry to ldrw with increments in BC
ldrw	281Ø	B	Line drawing subroutine, origin in COORDS, increments on calc stack
len_	3A8F	FF	Replaces params of A$ on calc stack with LEN A$
lend	1BØ9	B	Checks validity of address in NXTLIN at end of line run
let2	2F6D	B	Enters complete existing string as new string & reclaims old one
let_	2EBD	B	Assigns values to old (bit 1 FLAGX set) or new variables
lfar	2DØC	B	Looks through arguments of DEF FNs before searching VARS area
liad	16D6	B	Gives RAM address for line number (in HL, out HL)
lihl	211E	B	Gets second line number to HL for DELETE lines
linØ	132Ø	E	Returns line number in DE (from Ø)
lino	1324	B	Returns line number in DE for location HL
list	1545	B	Executes LIST command
lkup	Ø77C	F	Look-up routine for tokens or messages in file
llis	1541	B	Executes LLIST command (opens printer channel)
ln__	362E	FF	Replaces X on calc stack by LN X
lonz	3A96	FF	Loop on non-zero (like DJNZ) using BREG as counter; FP op 35H
lprn	2155	B	Executes LPRINT bv opening channel P first
lrun	1AØ6	B	The RUN entry point for the parser; 7 FLAGS is Z
lsnm	Ø4E8	C	A tape-name routine (?)
ltok	2543	B	Supplies the LOAD token in C
lvar	2C7Ø	B	Looks up variable pointed to by CH_ADD, NC if found, HL --> last letter in VARS
main	ØE28	B	Froduces automatic listing and waits for new line
memt	1FBB	B	Tests for top of usable memory and gives report 4 if insuff.
move	25DØ	B	Supplies MOVE token in B
msfi	ØF65	O	Error message file (ASCII with bit 7 of last char set)
msgs	Ø73F	F	Frints error messaoes
mtem	334A	FF	Executes A = 1Ø * A + C with carry returned in C
muli	3468	FF	Multiplies 16-bit integers: HL = HL * DE
mult	3489	FF	FP multiplication; uses integer multiple for small integers
n<Ø?	3921	FF	Tests calc stack last value, stacks 1 if negative, else Ø
n=m?	3956	FF	Ferforms 12 <=> comparisons between Nos. and strings (from calc stack)
n>Ø?	3914	FF	Tests last no. on calc stack & stacks 1 if positive, else Ø
neg_	382D	FF	FF op to change sign of last value on calc stack
new_	ØD1D	O	The BASIC NEW command (be careful)
nexl	1B27	B	Sets up NXTLIN from HL and goes into statement loop
next	1D55	B	Executes NEXT command; adds step to value & tests
nmir	ØØ66	O	Nonmaskable interrupt routine (has a bug)
nogo	38C5	FF	Wav out of cusr if no cartridge present
nonm	ØØ7Ø	O	Returns when no nonmaskable interrupt address (or should)
not_	391C	FF	Executes NOT: stacks 1 if last value is Ø, else stacks 1
nsin	3842	FF	Subroutine for ABS_ and NEG_ for small integers
numb	16Ø2	B	Skips floating point form if A holds ØE marker
nume	3ØD9	B	Returns NC if A holds a digit
nxch	ØØ74	O	Increments CH_ADD and puts character in A
nxli	165B	B	Fetches next line number into (HL) & (HL+1)

nxlo	1D84	B	Checks NEXT loop limit; sets C if done
oecn	2Ø8E	B	Executes ON ERR CONTINUE
oegt	2ØBC	B	Executes ON ERR GO TO
ono1	1788	B	Prints out number in BC up to 9999 for BASIC lines
ono2	1795	B	Prints no. pointed to by HL to 9999 for BASIC
ono3	179D	B	Prints no. in HL to four digits
ope2	1465	B	Gets channel from calc stack and opens channel
opeK	14CE	B	Open channel K (keyboard)
opeP	14D6	B	Open channel P (printer)
opeS	14D2	B	Open channel S (screen)
open	142A	B	Executes OPEN #N for channels K,S, & P
opfi	2B53	F	File correlating ASCII for arithmetic ops with ROM op codes for same
opid	1488	B	OPENs intellignet device
oppr	2ACB	B	Pushes function op code and priority onto machine stack
opty	2B31	B	Switches operator type when string op has priority over numeric
or __	3936	FP	Executes OR on two calc stack values
osfi	14C7	B	Offset table for open-stream routines
out_	1FØ4	B	OUT: gets values from stack and executes
pall	Ø6B4	P	Sends the character form to screen or printer
pank	23DE	B	Handles PAPER and INK routines (C set for INK)
pany	Ø63B	P	Print any characters subroutine
pars	1A27	B	The main BASIC parser; syntax-check entry point
pasb	25E4	O	Passes parameters to CALL_BANK routine
pasm	25B9	O	Passes parameters to bus expansion unit
pass	1E94	B	Passes over DATA or DEF FN during a run
paus	1FEB	B	Executes PAUSE command
pbas	1683	B	Prints characters and tokens in a BASIC line
pbl2	1671	B	Part of print-a-BASIC-line loop
pbln	1676	B	Print BASIC line no. specified by HL
pcch	Ø584	P	Handles control characters with operands (INK to OVER)
pcha	12B3	O	P channel (printer) flag set routine
pchr	Ø69A	P	Fetches character form from file pointed to by CHARS
pcht	Ø5FØ	P	Prints printable characters
pcom	Ø576	P	Print comma (tab) routine
pctr	2198	B	Prints various control characters: AT, TAB, color, expressions
pcur	162D	B	Prints C, E, G, K, or L cursor
peek	386B	fp	Replaces last value on calc stack by contents of that memory address
perc	1BF9	B	Makes temp colors permanent for color commands (Class 7)
plo2	263E	B	Subroutine to do actual PLOT; CALL with coords in BC
plot	2635	B	PLOT command; gets coords from calc stack and plots
poic	2624	B	For coords on calc stack, stacks Ø if color of paper, 1 if color of ink
poke	1FØA	B	POKE: gets values from calc stack and executes
pout	Ø5ØØ	P	Printout routine normally called by RST 1Ø
pqst	Ø58Ø	P	Prints question mark for unprintable codes
pra2	11ED	P	Prints character code in A
praa	11EA	P	Prints absolute value (A) as a character code
prat	Ø5B2	P	Print AT line & column in BC
prcr	2197	B	Prints a carriage return (ØD, CHR$(13))
prfi	2B6E	B	Priority table for arithmetic ops
prin	2159	B	PRINT routine; opens channel S, moves pointer from AROS
prn$	21DB	B	Print a string; BC holds length, DE points to start

prpr	Ø776	F	Prints characters recursively, saves registers
pseq	217E	B	Prints a sequence of characters whether to screen or printer
ptrs	12CA	E	Revises pointers after an insertion
pwr_	3C6C	FP	Raises last value on calc stack to power of next; continues at 3C78 as FP
pxad	26Ø3	B	Gives address of D-file byte in HL, pixel as A-7, for coordinates in BC
qrem	3ABB	FP	Replaces X and Y on calc stack by their quotient (last val) and remainder
quot	2971	B	Handles quotes with strings and VAL$ and embedded quotes
rafp	3761	FP	Takes contents of A and runs corresponding FP op for BASIC interpreter
rall	246Ø	O	Does cold-start reset of all devices
rand	1ED4	B	Executes RANDOMIZE to set SEED
rdin	3A6Ø	FP	Reads in character from channel (Ø-15) specified on calc stack
read	1D97	B	Executes READ command
rec1	174D	B	Reclaims memory from DE to HL - 1
rec2	175Ø	B	Reclaims BC bytes from HL onward
rem_	1BØØ	B	Executes BASIC REM; ignores rest of line
res2	3652	FP	Restacks two small integers in fp form
ress	3655	FP	Subroutine for res2, so the routine runs twice
retn	1FD4	B	Executes RETURN; gets line and statement no. from GOSUB stack
roun	35D3	FP	An fp op to truncate a number toward zero to integer form
rres	1ECA	B	Used by RUN to do a RESTORE
rsØ8	ØØØ8	O	BASIC error trap; breaks to print message
rs1Ø	ØØ1Ø	O	Sends character in A to screen or printer
rs18	ØØ18	O	Gets next printable character at CH_ADD or above to A
rs2Ø	ØØ2Ø	O	Increments CH_ADD and gets next printable character
rs28	ØØ28	O	Jumps to floating-point calculator mode
rs3Ø	ØØ3Ø	O	Creates BC spaces in BASIC workspace (WORKSP)
rs38	ØØ38	O	Increments clock and scans keyboard (6Ø times/sec)
rse2	247F	O	Checks whether RESET specifies a single device
rse3	2498	O	Gets stream data to DE and resets intelligent device
rset	2ØAE	B	Executes ON ERR RESET
rsew	2487	O	Does warm start of all current devices
rsrv	132D	O	Opens workspace below the calculator stack (for RST 3Ø)
rsta	3656	FP	FP op to send the number pointed to by HL to calc stack
rstr	1E9D	B	Executes RESTORE command
run_	1F2B	B	Executes the RUN command
runt	2B22	B	Records numeric or string in FLAGS bit 6
rusr	3882	FP	Return routine for USR when cartridge is present
s$el	2DEA	B	Gets parameters of string array element to calc stack
s-fn	2AAB	B	Expression scan for functions CODE (AF) to NOT (C3)
sNot	2ABØ	B	Expression scan for NOT
saln	2A42	B	Expression scan for alphanumeric character
sano	2DEØ	B	Sets HL to point one before floating point bytes of array element
sapp	38Ø8	FP	Series approximator for calculating transcendentals (SIN, EXP, etc)
sarr	2D6C	B	Gets array dimension to B, separates numeric and string arrays
satr	2A3Ø	B	Expression scan for ATTR
satt	Ø71Ø	F	Sets and stores attribute byte for printed character
sbin	2A4B	B	Expression scan for decimal number or for BIN
sc$2	2891	B	Entry point to read screen with coords in BC (col/line)
sca2	2ADØ	B	Continues expression scan for further subexpressions
scha	12A8	O	S channel (main screen) flag set routine
scl2	Ø93B	F	Scrolling subroutine; no. lines in B

sclo	2AF2	B	Scan loop to evaluate nested functions by their priority
scng	2854	B	Scans and evaluates expressions, puts result on calc stack
scr$	288E	B	Returns character on screen at coords from calc stack
scr2	Ø83D	F	Handles lower screen after a scroll
scrl	Ø939	F	Scrolling subroutine for 23-line scroll
scro	Ø8ØD	F	Scrolls the display
sdfi	11C1	F	Initial stream data file
sdfn	2BB5	B	Searches for a DEF FN in program to evaluate FN
sdfp	Ø914	F	Set display file parameters from BC (top left = 1821)
sele	2DA5	B	Finds parameters of an array element
sepa	1AB2	B	Checks for proper separator and faults to error C
sest	1354	E	Clears calc stack
sffi	294C	F	Offset table for expression scanning functions and operators
sgnm	3851	FP	SGN op; returns 1 on calc stack for +, Ø for Ø, -1 for -
shif	339C	FP	Shifts an fp form right to line up for addition
sine	3BDØ	FP	Replaces X on calc stack with SIN X
sint	314A	B	Stores small integer (- to +65535) at (HL) and next 4 bytes
skfn	2C69	B	Skips over characters in DEF FN without changing CH_ADD
skip	ØØ7D	O	Sorts and skips nonprintable characters for RST18/2Ø
skpt	2569	O	Reads through a statement in applications cartridge
slet	2A87	B	Scans for letter, looks up variable, stacks it on calc stack
slic	2E1Ø	B	Main handler for string slicing
slug	ØDØD	E	Removes floating-point forms from BASIC lines
smdt	14ØF	B	Gets stream data to BC
smin	133F	E	Clears edit area, workspace, and calc stack
sneg	2A9D	B	Expression scan for minus sign
snex	1B4B	B	Checks whether next statement or next line follows
snum	3773	FP	Moves FP form to calc stack from elsewhere in memory
soun	2128	B	Executes SOUND command
spcf	2179	B	Sets flag to print copyright & curley brackets
spnt	39DA	FP	Calc stack pointer set: HL to last value, DE to next
spoi	2A39	B	Expression scan for POINT
sqrt	3C65	FP	Replaces X on calc stack with SQR X
srnd	29B6	B	Calculates RND from SEED
sscr	2A26	B	Expression scan for SCREEN$
ssli	2D96	B	Looks for a slicer subscript in handling string arrays
sst$	2AC5	B	Expression scan for STR$ and for CHR$
stak	37DA	FP	Stacks one of the constants (Ø,1,.5,pi/2,1Ø) according to 2nd nibble
stbc	3ØE9	B	Puts absolute value in BC on calc stack (Ø-65535)
stda	37A7	FP	Gets data to calc stack as new FP number
stde	ØCFB	E	Sets DE to end of workspace (WORKSP)
stdg	3ØEØ	B	If A holds a digit, that digit goes onto calc stack
stfi	Ø4AC	F	Semitone data file, 5 nos. per tone
stfp	3785	FP	Stacks fp form of a number supplied in code following op 34
sthl	ØCF6	E	Sets HL to start of workspace
stiØ	29A8	B	Jump to the STICK routine
sti1	28F8	B	Routine for the STICK command; checks initial parameters
sti2	2926	B	Checks for button pushed/unpushed
stik	29Ø2	B	Main routine for STICK
stk$	2D5F	B	Stacks parameters for a simple string from VARS area
stk5	2E74	B	Sends AEDCB to calc stack
stka	3ØE6	B	Puts absolute value in A onto calc stack (Ø-255)
stkv	2D54	B	Finds string parameters or address of array element (HL) in VARS

stmt	1A44	B	Subroutine for evaluating statements in a line
stok	253F	B	Supplies the SAVE token in C
stor	37EC	FF	Moves FF form from calc stack to MEM slot (storØ to stor5)
stp2	Ø613	P	Stores updated print position (lower screen)
stp3	Ø613	P	Stores updated printer buffer variables
stpo	Ø5F3	P	Stores the updated print position (upper screen)
str$	3A3A	FF	Replaces X on calc stack by params of STR$ X
strt	1AB9	B	Return point after every statement, checks BREAK
stup	251E	O	Setup to send tokens for disklike commands to bus expansion unit
sub_	33CE	FF	Subtract routine; changes a sign and proceeds to add_
sudf	2B7B	B	Scan to evaluate user defined functions
svl$	2AA4	B	Expression scan for VAL$
swop	37FB	FF	Exchanges the order of last two FP forms on calc stack
swor	134E	E	Clears workspace and calc stack
syns	214F	B	Escape routine for syntax checking
synt	2B19	B	Syntax test to insure numbers for arithmetic ops, strings for string ops
synz	2889	B	Tests the syntax-checking flag
szer	37BØ	FF	Adds zeroes to calc stack to fill out FP form
tan_	3BF5	FF	Replaces X on calc stack with TAN X
tchk	238Ø	B	Routine to check for channel K (lower screen) in use
tes5	3768	FF	Tests for 5 bytes more of memory for a new FP form
tesk	Ø35C	O	Tests key value and gets main code from kyfi
texp	362B	FF	Tests exponent for large numbers; subroutine for roun
tofi	ØØ98	P	BASIC token name file (ASCII w. bit 7 set for last char)
toks	Ø745	P	Expands and prints BASIC tokens
tost	2A73	B	Routine to stack (calc stac) a numeric result from scan
tovr	2F64	B	Passes numbers from stack & strings from workspace to VARS area
tpar	287B	B	Tests for parens with two parameters enclosed, stacks them
tpfi	3C8A	F	File of ASCII cassette messages
tquo	2668	B	Tests for closing quotes in an expression
trsp	Ø77Ø	P	Prints trailing space after token
ts12	292B	B	Tests for a 1 or 2 in A; gives error A otherwise; for STICK
tsc2	Ø7C3	P	Tests whether the 'scroll?' prompt is needed
tsco	Ø79Ø	P	Tests whether scroll is necessary
upls	296D	B	Unary plus routine skips over to next character and to scan
usbc	266Ø	FF	Unstack BC; last calc stack value to B, next last to C, signs to DE
usr#	3872	FF	Executes USR X, where X is last value on calc stack
usr$	38D7	FF	Executes USR$ from string parameters on calc stack
usta	266D	FF	Gets last value (Ø-255) on calc stack to A, sign to C
uzro	1C51	B	Puts a zero on calc stack for commands like RUN
val$	39F9	FP	Handles both VAL and VAL$, returns no. on calc stack
zert	39Ø4	FP	Tests FF form pointed to by HL for Ø, returns C set if so

RAM RESIDENT CODE

BANO	645E	RR	Gets bank no. for addr HL into A
BAST	6405	RR	Gets bank status of bank B into B, horiz select to C
BMAP	66E8	RR	Creates bit map for active chunks, start addr in HL
BSST	651E	RR	Puts status of all banks on stack as pointed to by IX
CALN	6274	RR	Executes function call after stack fix up
CBAN	65D0	RR	CALL bank; horiz select & addr from stack plus params in & out
CHUN	644D	RR	Gets chunk for addr HL from high 3 bits of H
ENAB	6499	RR	Enables bank B, horiz select C
FUNC	6200	RR	Function dispatcher for mc users; JP works, CALL crashes
GBAN	6572	RR	Goto bank; horiz select & addr from stack, no return
GOEX	6615	RR	Goes to HL in EXROM
GWOR	6316	RR	Gets 16-bit word at addr HL bank B into HL
MOVB	668C	RR	Moves DE bytes from bank to bank, direction in A
PWOR	633B	RR	Puts DE at address HL in bank B
RBSR	63AD	RR	Reads bank status reg described nibblewise by DE, returns data in E
RBST	654A	RR	Restores bank status from stack as pointed to by IX
WBSR	635C	RR	Writes E to bank status register in D
XFER	6722	RR	Main routine for transfers from bank to bank
XINT	62AE	RR	Fields RST 38 interrupt while EXROM is resident
XNMI	6307	RR	Would handle NMI but for JR NZ bug and lack of connecting code

RAM-RES (Numeric)

FUNC	6200	RR	Function dispatcher for mc users; JP works, CALL crashes
CALN	6274	RR	Executes function call after stack fix up
XINT	62AE	RR	Fields RST 38 interrupt while EXROM is resident
XNMI	6307	RR	Would handle NMI but for JR NZ bug and lack of connecting code
GWOR	6316	RR	Gets 16-bit word at addr HL bank B into HL
PWOR	633B	RR	Puts DE at address HL in bank B
WBSR	635C	RR	Writes E to bank status register in D
RBSR	63AD	RR	Reads bank status reg described nibblewise by DE, returns data in E
BAST	6405	RR	Gets bank status of bank B into B, horiz select to C
CHUN	644D	RR	Gets chunk for addr HL from high 3 bits of H
BANO	645E	RR	Gets bank no. for addr HL into A
ENAB	6499	RR	Enables bank B, horiz select C
BSST	651E	RR	Puts status of all banks on stack as pointed to by IX
RBST	654A	RR	Restores bank status from stack as pointed to by IX
GBAN	6572	RR	Goto bank; horiz select & addr from stack, no return
CBAN	65D0	RR	CALL bank; horiz select & addr from stack plus params in & out
MOVB	668C	RR	Moves DE bytes from bank to bank, direction in A
BMAP	66E8	RR	Creates bit map for active chunks, start addr in HL
XFER	6722	RR	Main routine for transfers from bank to bank
GOEX	6615	RR	Goes to HL in EXROM

EXROM NAMES

akey	Ø8AA	O	Waits for a keystroke
aro?	Ø9ØF	O	Checks for applications cartridge and jumps if there
asig	ØBD1	O	Assigns bank number to current bank
bood	Ø99A	O	Boots highest priority device
boot	ØØ5A	O	Sets up xout at 6ØØØ as boot routine for BASIC ROM
bsct	Ø9F4	O	Builds current system configuration table
cbnk	ØF99	O	Call a routine in another bank
cent	Ø1AB	C	Cassette op entry routine; op is in taddr; sorts for syntax
chir	ØC1F	O	Marks intelligent devices and initializes if initializable
cidi	ØC2F	O	Calls intelligent device initialization routine
cld2	ØE27	V	Closes DFILE2 and clears video mode
edge	Ø18D	C	Counts and times pulse edges during LOAD and VERIFY
erro	ØØØ8	O	Error interrupt handler
exin	Ø8E7	O	Initialization check for cartridge
fun1	1FD8	F	Jump table for RAM-res code; half wrong by one byte
fun2	1FEC	F	Jump table for functions in EXROM; use data and EXROM NAMEs
funf	1EDC	F	Jump table for functions in ROM; use data mode and ROM NAMEs
jbnk	ØF6A	O	Jump interbank
lang	Ø91F	O	Tests for cartridge language
lblo	Ø5C6	C	Loads a block of bytes and returns
ldby	ØØFC	C	Subroutine to LOAD bytes from tape
load	Ø5CC	C	Control routine for LOAD
lro?	Ø8FØ	O	Checks for presence of language cartridge and jumps to it
melv	Ø7E8	C	MERGE a line or variable
merg	Ø6E5	C	Control routine for MERGE
mlst	Ø928	O	Machine language start up for cartridge
nova	Ø76C	O	Initializes SVs without leaving space for ml variables
nram	ØADB	O	Test a new bank for RAM, moves in keyboard interrupt handler
opd2	ØDBØ	V	Opens DFILE2 and sets video mode
pass	ØF43	O	Passes characters via bus expansion unit
rebo	ØØE5	C	Restores border color at end of a cassette op
rnob	ØCFB	O	Renumbers expansion banks in order of interrupt priorities
rset	ØC4C	Ø	Performs RESET command on bus expansion unit
save	Ø851	C	Control routine for SAVE
sbas	Ø956	O	Starts BASIC applications cartridge
svby	ØØ68	C	Subroutine to SAVE bytes to tape
svid	ØE8E	V	Switches video mode per value in VIDMOD
veri	Ø58F	C	Control routine for VERIFY
vtab	1DØØ	F	Table for fixing up addresses when RAM-res code is moved high
xini	ØØ49	O	Initializer; enables all of home bank excpt chunk Ø
xout	ØØ4F	O	Disables and exits EXROM
xr38	ØØ38	O	Fields keyboard/clock interrupt when EXROM is in
xxxx	1ØØØ	O	ROM copy of RAM resident code; gets moved to 62ØØH

TS 2Ø68 ROM ADDRESSES

rsØ8	ØØØ8	O	BASIC error trap; breaks to print message
rs1Ø	ØØ1Ø	O	Sends character in A to screen or printer
rs18	ØØ18	O	Gets next printable character at CH_ADD or above to A
rs2Ø	ØØ2Ø	O	Increments CH_ADD and gets next printable character
rs28	ØØ28	O	Jumps to floating-point calculator mode
rs3Ø	ØØ3Ø	O	Creates BC spaces in BASIC workspace (WORKSP)
rs38	ØØ38	O	Increments clock and scans keyboard (6Ø times/sec)
ertr	ØØ53	O	Fetches error no. to ERR_NR & resets stack
nmir	ØØ66	O	Nonmaskable interrupt routine (has a bug)
nonm	ØØ7Ø	O	Returns when no nonmaskable interrupt address (or should)
nxch	ØØ74	O	Increments CH_ADD and puts character in A
skip	ØØ7D	O	Sorts and skips nonprintable characters for RST18/2Ø
tofi	ØØ98	F	BASIC token name file (ASCII w. bit 7 set for last char)
kyfi	Ø227	F	Key tables for interpretation of keyboard modes.
kbsc	Ø2BØ	O	Keyboard scan, returns Ø-39d in E, shift state in D
keyn	Ø2E1	O	Main keyboard read and decode; key to LAST_K, set 5, FLAGS
kend	Ø32E	O	End of keyn routine if a key pressed
krep	Ø336	O	Repeating key routine; sorts for tokens and DELETE
tesk	Ø35C	O	Tests key value and gets main code from kyfi
deck	Ø371	O	Decodes key value according to mode and shift state
bper	Ø3F3	S	Beeps notes according to values in DE & HL. Callable.
beep	Ø436	S	Beeps in pitch and duration from calc stack (2 nos.)
stfi	Ø4AC	F	Semitone data file, 5 nos. per tone
lsnm	Ø4E8	C	A tape-name routine (?)
pout	Ø5ØØ	P	Printout routine normally called by RST 1Ø
ccfi	Ø528	P	Table of offsets for control-character subroutines
klft	Ø53A	P	Cursor-left routine
krgt	Ø554	P	Cursor-right routine
cret	Ø566	P	Carriage-return routine
pcom	Ø576	P	Print comma (tab) routine
pqst	Ø58Ø	P	Prints question mark for unprintable codes
pcch	Ø584	P	Handles control characters with operands (INK to OVER)
prat	Ø5B2	P	Print AT line & column in BC
pcht	Ø5FØ	P	Prints printable characters
stpo	Ø5F3	P	Stores the updated print position (upper screen)
stp2	Ø613	P	Stores updated print position (lower screen)
stp3	Ø613	P	Stores updated printer buffer variables
gtpo	Ø61A	P	Get current print position parameters
gtp2	Ø634	P	Get current printer position parameters
pany	Ø63B	P	Print any characters subroutine
pchr	Ø69A	P	Fetches character form from file pointed to by CHARS
pall	Ø6B4	P	Sends the character form to screen or printer
satt	Ø71Ø	P	Sets and stores attribute byte for printed character
msgs	Ø73F	P	Prints error messages
toks	Ø745	P	Expands and prints BASIC tokens
trsp	Ø77Ø	P	Prints trailing space after token
prpr	Ø776	P	Prints characters recursively, saves registers
lkup	Ø77C	P	Look-up routine for tokens or messages in file
tsco	Ø79Ø	P	Tests whether scroll is necessary
tsc2	Ø7C3	P	Tests whether the 'scroll?' prompt is needed
scro	Ø8ØD	P	Scrolls the display
scr2	Ø83D	P	Handles lower screen after a scroll

ctem	Ø888	P	Sets temporary color values
cls_	Ø8A6	P	Executes BASIC CLS; callable
clow	Ø8A9	P	Clears lower screen (command lines)
clds	Ø8EA	P	Subroutine to clear display
sdfp	Ø914	P	Set display file parameters from BC (top left = 1821)
scrl	Ø939	P	Scrolling subroutine for 23-line scroll
scl2	Ø93B	P	Scrolling subroutine; no. lines in B
clli	Ø97F	P	Clears lower B lines of the display
gatr	Ø9C3	P	Returns attribute address (DE) for given display addr (HL)
ladr	Ø9D6	P	Gives display address (HL) for screen line (B)
delk	Ø9E7	?	Delays and waits for a keystroke (use unknown)
copy	ØAØ2	P	BASIC COPY command (callable)
cbuf	ØA23	P	Sends contents of printer buffer to printer
clpb	ØA35	P	Clears the printer buffer
cpli	ØA4A	P	Copies one pixel line to printer
edtr	ØA82	E	Editor for BASIC line entry or INPUT
adch	ØAE7	E	Adds a character to EDIT or INPUT line
edky	ØAF8	E	Handles edit keys during line entry
edfi	ØBØ6	E	Offset table for edit-key subroutines
edit	ØBØF	E	Handles EDIT key functions, including INPUT
eddn	ØB59	E	Handles cursor-down during edit
edst	ØB67	E	Handles STOP key during INPUT
edlf	ØB6D	E	Handles cursor-left during EDIT
edrt	ØB72	E	Handles cursor-right during EDIT
eddl	ØB7B	E	Handles DELETE during EDIT
edot	ØB84	E	Reads & ignores 2 characters and ends edit in error
edlm	ØB97	E	Moves cursor toward start of edit-line
edup	ØBBF	E	Handles cursor-up during edit
edgr	ØBDC	E	Handles graphics codes during EDIT
eder	ØBE5	E	Handles errors during EDIT
clws	ØBFD	E	Clears the editing workspace
ksca	ØC15	E	Scans keyboard and returns keycode (Try it)
echp	ØC83	E	Echoes keyboard buffer to current channel (lower screen)
sthl	ØCF6	E	Sets HL to start of workspace
stde	ØCFB	E	Sets DE to end of workspace (WORKSP)
slug	ØDØD	E	Removes floating-point forms from BASIC lines
new_	ØD1D	O	The BASIC NEW command (be careful)
init	ØD31	O	Main initialization routine when 2Ø68 is switched on
main	ØE28	B	Produces automatic listing and waits for new line
msfi	ØF65	O	Error message file (ASCII with bit 7 of last char set)
basl	1158	E	Adds a new BASIC line to existing program
cnfi	11AA	F	Initial channel address file
sdfi	11C1	F	Initial stream data file
gkey	11CF	E	Gets keyboard input during INPUT and EDIT
inpa	11E1	E	Saves registers and points HL to input address
praa	11EA	P	Prints absolute value (A) as a character code
pra2	11ED	P	Prints character code in A
chop	123Ø	O	Channel-open routine (FD-Ø3 as stream no. in A)
cho2	123F	O	Find address of channel for given stream
chfl	124D	O	Set flags for channel
chex	1265	O	Channel exchange routine

chfi	1293	O	Channel-code offset table
kcha	129A	O	K channel (lower screen) flag set routine
scha	12A8	O	S channel (main screen) flag set routine
pcha	12B3	O	P channel (printer) flag set routine
1spa	12B8	E	Opens one space at area designated by HL
cins	12BB	E	Opens BC spaces at address HL
ptrs	12CA	E	Revises pointers after an insertion
linØ	132Ø	E	Returns line number in DE (from Ø)
lino	1324	B	Returns line number in DE for location HL
rsrv	132D	O	Opens workspace below the calculator stack (for RST 3Ø)
smin	133F	E	Clears edit area, workspace, and calc stack
swor	134E	E	Clears workspace and calc stack
sest	1354	E	Clears calc stack
celi	1363	E	Clears edit line
indx	136B	O	Indexes into tables for various look-ups
fdev	1374	O	Searches config table for device spec in C
clos	139F	B	Executes CLOSE #N (closes channel)
clo2	13BE	B	Closes channel with channel address BC
clo3	13D8	B	Closes intelligent device
csfi	14Ø7	B	Table of offsets for close stream routines
clsm	14ØD	B	Fetches channel pointer for close-stream routine
smdt	14ØF	B	Gets stream data to BC
open	142A	B	Executes OPEN #N for channels K,S, & P
ope2	1465	B	Gets channel from calc stack and opens channel
opid	1488	B	OPENs intellignet device
osfi	14C7	B	Offset table for open-stream routines
opeK	14CE	B	Open channel k (keyboard)
opeS	14D2	B	Open channel S (screen)
opeP	14D6	B	Open channel P (printer)
Blis	14E1	B	List the BASIC program to screen
llis	1541	B	Executes LLIST command (opens printer channel)
list	1545	B	Executes LIST command
blin	15A1	B	Prints a BASIC line for the LIST command
numb	16Ø2	B	Skips floating point form if A holds ØE marker
flas	16ØD	B	Prints flashing cursors
pcur	162D	B	Prints C, E, G, K, or L cursor
nxli	165B	B	Fetches next line number into (HL) & (HL+1)
pbl2	1671	B	Part of print-a-BASIC-line loop
pbln	1676	B	Print BASIC line no. specified by HL
pbas	1683	B	Prints characters and tokens in a BASIC line
liad	16D6	B	Gives RAM address for line number (in HL, out HL)
clno	16E8	B	Compares line no. in BC with (HL), returns Z for match
fist	16FØ	B	Finds statement D in a BASIC line (or token E)
adnx	172Ø	B	Finds address of next program line or next variable
diff	1745	B	Sets BC = HL - DE; returns HL & DE exchanged
rec1	174D	B	Reclaims memory from DE to HL - 1
rec2	175Ø	B	Reclaims BC bytes from HL onward
elno	1768	B	Gets line number of line in edit area to BC
ono1	1788	B	Prints out number in BC up to 9999 for BASIC lines
ono2	1795	B	Prints no. pointed to by HL to 9999 for BASIC
ono3	179D	B	Prints no. in HL to four digits

arin	17B5	O	Bankswitches for cartridge software (BASIC)
arln	17CF	O	Searches for line no. BC in cartridge
aros	18C6	O	Sets up buffer for cartridge software
bcfi	1945	B	BASIC command routine offset table
pars	1A27	B	The main BASIC parser; syntax-check entry point
stmt	1A44	B	Subroutine for evaluating statements in a line
sepa	1AB2	B	Checks for proper separator and faults to error C
strt	1AB9	B	Return point after every statement, checks BREAK
lrun	1AD8	B	The RUN entry point for the parser; 7 FLAGS is Z
fnwl	1AEC	B	Finds new line address after a program jump
rem_	1BØØ	B	Executes BASIC REM; ignores rest of line
lend	1BØ9	B	Checks validity of address in NXTLIN at end of line run
nexl	1B27	B	Sets up NXTLIN from HL and goes into statement loop
cknd	1B44	B	Syntax check routine; faults to error unless at line end
snex	1B4B	B	Checks whether next statement or next line follows
clfi	1B64	B	Command class routine offset table
clØ3	1B7Ø	B	Class 3: RUN, RAND, CLEAR, RESTORE
clØØ	1B73	B	Class Ø; STOP, RETURN, NEW, CONT, CLS, COPY
clØ5	1B74	B	Class 5: DEF FN,DELETE,ON ERR,RESET,SOUND
clØ5	1B74	B	Class 5: PRINT,INPUT,DIM,REM,LIST,READ,DATA,LPRINT,LLIST
comr	1B79	B	Gets command routine address from syntax table and jumps
clØ1	1B82	B	Class 1: LET
clØ2	1BB1	B	Assigns value to variable in LET statement
gval	1BB9	B	Evaluates expression to get value for LET or READ
gva2	1BBC	B	Evaluates expression to get value for INPUT
clØ4	1BCF	B	Class 4: FOR, NEXT command routines
2num	1BDD	B	Class 8: POKE, BEEP, OUT
2num	1BDD	B	Evaluates two expressions for Class 8 commands
1num	1BE5	B	Class 6: GOTO,IF,GOSUB,PAUSE,BORDER,OPEN,CLOSE
1num	1BE5	B	Evaluate one expression for command class 6
g$tr	1BEF	B	Class ØA: FORMAT,MOVE,ERASE,CAT
g$st	1BEF	B	Evaluate string expression for command class ØA
perc	1BF9	B	Makes temp colors permanent for color commands (Class 7)
clØ9	1C29	B	Class 9: PLOT,DRAW,CIRCLE; sets default conditions
clØb	1C46	B	Class ØB: cassette routines
fnum	1C49	B	Fetches number if there else puts zero on stack
uzro	1C51	B	Puts a zero on calc stack for commands like RUN
Stop	1C59	B	Error 9 trap for STOP command
if _	1C5B	B	Executes IF command on last calc stack value
for _	1C76	B	Executes FOR command with value and limit on calc stack
fito	1D26	B	Finds match for token in E starting at (HL)
next	1D55	B	Executes NEXT command; adds step to value & tests
nxlo	1D84	B	Checks NEXT loop limit; sets C if done
read	1D97	B	Executes READ command
data	1E62	B	DATA statement; syntax gets checked, but as REM in run
pass	1E94	B	Passes over DATA or DEF FN during a run
rstr	1E9D	B	Executes RESTORE command
rres	1ECA	B	Used by RUN to do a RESTORE
rand	1ED4	B	Executes RANDOMIZE to set SEED
cont	1EE4	B	CONTINUE: loads up line and statement no. for jump
goto	1EF1	B	GOTO: gets and tests line number for jump

out_	1FØ4	B	OUT: gets values from stack and executes
poke	1FØA	B	POKE: gets values from calc stack and executes
get2	1FØF	B	Gets two values from calc stack to A and BC
1int	1F1E	B	Gets 1-byte integer from calc stack to A
2int	1F23	B	Gets 2-byte integer from calc stack to BC
run_	1F2B	B	Executes the RUN command
cler	1F36	B	Executes the CLEAR routine
clrn	1F39	B	Entry point to CLEAR used by RUN
gosb	1F99	B	Executes the GOSUB command
memt	1FBB	B	Tests for top of usable memory and gives report 4 if insuff.
retn	1FD4	B	Executes RETURN; gets line and statement no. from GOSUB stack
paus	1FEB	B	Executes PAUSE command
brek	2ØØ9	O	Reads BREAK key; returns NC if SHIFT-BREAK is pressed
dffn	2Ø1D	B	DEF FN command; check for syntax, skipped in RUN
oecn	2Ø8E	B	Executes ON ERR CONTINUE
rset	2ØAE	B	Executes ON ERR RESET
oegt	2ØBC	B	Executes ON ERR GO TO
dell	2ØD1	B	Executes DELETE (lines) command
lihl	211E	B	Gets second line number to HL for DELETE lines
soun	2128	B	Executes SOUND command
syns	214F	B	Escape routine for syntax checking
lprn	2155	B	Executes LPRINT by opening channel P first
prin	2159	B	PRINT routine; opens channel S, moves pointer from AROS
spcf	2179	B	Sets flag to print copyright & curley brackets
pseq	217E	B	Prints a sequence of characters whether to screen or printer
prcr	2197	B	Prints a carriage return (ØD, CHR$(13))
pctr	2198	B	Prints various control characters: AT, TAB, color, expressions
prn$	21DB	B	Print a string; BC holds length, DE points to start
endp	21E4	B	End of print; tests for), carriage ret, and colon
ctch	21ED	B	Handles position control characters in PRINT: simicolon, comma, apostrophe
#stm	22ØF	B	Routine to change active stream
inpt	222B	B	Main input routine; opens channel K
inpr	226B	B	Handles control items during INPUT
inpl	2282	B	Handles INPUT LINE
inps	2297	B	Handles simple input variables
iprm	22A4	B	Put INPUT prompt into workspace, gets input and assigns it
inas	2363	B	Subroutine to assign an INPUT value to a variable
inst	237A	B	Handles STOP in an INPUT line
tchk	238Ø	B	Routine to check for channel K (lower screen) in use
col1	238B	B	Gets next character to sort for color controls
col2	238C	B	Sorts for color item followed by semicolon or comma
col3	239C	B	Subroutine to sort for INK,PAPER,FLASH,BRIGHT,INVERSE,OVER
col4	23A6	B	Reduces color token to control character and sends to screen
colv	23BB	B	Sets color system variables for PRINT
pank	23DE	B	Handles PAPER and INK routines (C set for INK)
cocl	2416	B	Changes a color system variable according to mask in B
brfl	241D	B	Handles BRIGHT and FLASH (C set for FLASH)
brdr	243E	B	BORDER command routine; gets color from calc stack, sets INK
brds	2441	B	Call-in point to set border with color in A (used by HZ)
crst	2454	B	Checks for cold start symbol after RESET
rall	246Ø	O	Does cold-start reset of all devices

rse2	247F	O	Checks whether RESET specifies a single device
rsew	2487	O	Does warm start of all current devices
rse3	2498	O	Gets stream data to DE and resets intelligent device
cass	24D2	B	Handles cassette commands for cassette or disklike devices
stup	251E	O	Setup to send tokens for disklike commands to bus expansion unit
stok	253F	B	Supplies the SAVE token in C
ltok	2543	B	Supplies the LOAD token in C
casr	2548	B	Does bank switch to EXROM for cassette routines
skpt	2569	O	Reads through a statement in applications cartridge
pasm	25B9	O	Passes parameters to bus expansion unit
cat_	25C8	B	Supplies CAT token in B
form	25CC	B	Supplies FORMAT token in B
move	25DØ	B	Supplies MOVE token in B
eras	25D4	B	Supplies ERASE token in B
pasb	25E4	O	Passes parameters to CALL_BANK routine
pxad	26Ø3	B	Gives address of D-file byte in HL, pixel as A-7, for coordinates in BC
poic	2624	B	For coords on calc stack, stacks Ø if color of paper, 1 if color of ink
plot	2635	B	PLOT command; gets coords from calc stack and plots
plo2	263E	B	Subroutine to do actual PLOT; CALL with coords in BC
usbc	266Ø	FF	Unstack BC; last calc stack value to B, next last to C, signs to DE
usta	266D	FF	Gets last value (Ø-255) on calc stack to A, sign to C
circ	2679	B	CIRCLE command routine
draw	26DB	B	DRAW command routine (26FC resumes floating point ops)
darc	2792	B	Arc-drawing subroutine
cdpm	27D6	B	Subroutine to set initial parameters for CIRCLE and DRAW
ldrw	281Ø	B	Line drawing subroutine, origin in COORDS, increments on calc stack
ldr2	2813	B	Use as entry to ldrw with increments in BC
scng	2854	B	Scans and evaluates expressions, puts result on calc stack
tquo	2868	B	Tests for closing quotes in an expression
tpar	287B	B	Tests for parens with two parameters enclosed, stacks them
synz	2889	B	Tests the syntax-checking flag
scr$	288E	B	Returns character on screen at coords from calc stack
sc$2	2891	B	Entry point to read screen with coords in BC (col/line)
fiat	28D7	B	Finds attribute at screen coords from calc stack, stacks attribute
dele	28ED	B	Handles DELETE key
sti1	28F8	B	Routine for the STICK command; checks initial parameters
stik	29Ø2	B	Main routine for STICK
sti2	2926	B	Checks for button pushed/unpushed
ts12	292B	B	Tests for a 1 or 2 in A; gives error A otherwise; for STICK
free	2934	B	Executes FREE statement
sffi	294C	F	Offset table for expression scanning functions and operators
freØ	2969	B	Jumps to main routine for FREE
stiØ	296B	B	Jump to the STICK routine
upls	296D	B	Unary plus routine skips over to next character and to scan
quot	2971	B	Handles quotes with strings and VAL$ and embedded quotes
brck	29A6	B	Gets closing bracket and loop to expression scan
srnd	29B6	B	Calculates RND from SEED
gtpi	29E5	B	Puts PI on calc stack
ink$	29F2	B	Executes INKEY$; stacks input string or empty string
sscr	2A26	B	Expression scan for SCREEN$
satr	2A3Ø	B	Expression scan for ATTR

spoi	2A39	B	Expression scan for POINT
saln	2A42	B	Expression scan for alphanumeric character
sbin	2A4B	B	Expression scan for decimal number or for BIN
tost	2A73	B	Routine to stack (calc stac) a numeric result from scan
slet	2A87	B	Scans for letter, looks up variable, stacks it on calc stack
sneg	2A9D	B	Expression scan for minus sign
svl$	2AA4	B	Expression scan for VAL$
s-fn	2AAB	B	Expression scan for functions CODE (AF) to NOT (C3)
sNot	2ABØ	B	Expression scan for NOT
sst$	2AC5	B	Expression scan for STR$ and for CHR$
oppr	2ACB	B	Pushes function op code and priority onto machine stack
sca2	2ADØ	B	Continues expression scan for further subexpressions
sclo	2AF2	B	Scan loop to evaluate nested functions by their priority
synt	2B19	B	Syntax test to insure numbers for arithmetic ops, strings for string ops
runt	2B22	B	Records numeric or string in FLAGS bit 6
opty	2B31	B	Switches operator type when string op has priority over numeric
opfi	2B53	F	File correlating ASCII for arithmetic ops with ROM op codes for same
prfi	2B6E	B	Priority table for arithmetic ops
sudf	2B7B	B	Scan to evaluate user defined functions
sdfn	2BB5	B	Searches for a DEF FN in program to evaluate FN
cpfn	2BD2	B	Compares found DEF FN with FN under evaluation
fnev	2BEF	B	Evaluates arguments of an FN using found DEF FN during scan
fnva	2C4B	B	Evaluates FN from argument values determined with DEF FN
skfn	2C69	B	Skips over characters in DEF FN without changing CH_ADD
lvar	2C7Ø	B	Looks up variable pointed to by CH_ADD, NC if found, HL --> last letter in VARS
lfar	2DØC	B	Looks through arguments of DEF FNs before searching VARS area
stkv	2D54	B	Finds string parameters or address of array element (HL) in VARS
stk$	2D5F	B	Stacks parameters for a simple string from VARS area
sarr	2D6C	B	Gets array dimension to B, separates numeric and string arrays
ssli	2D96	B	Looks for a slicer subscript in handling string arrays
sele	2DA5	B	Finds parameters of an array element
sano	2DEØ	B	Sets HL to point one before floating point bytes of array element
s$el	2DEA	B	Gets parameters of string array element to calc stack
slic	2E1Ø	B	Main handler for string slicing
$stk	2E6F	B	Stacks parameters for a sliced or array-element string
stk5	2E74	B	Sends AEDCB to calc stack
cpit	2E8A	B	Evaluates next expression, compares with limit in HL, gives A=FF if over, else Ø
de+1	2EAC	B	Loads (DE+1) to DE, points HL to DE+2
hl*d	2EB2	B	Sets HL= HL*DE; gives error 4 if overflow
let_	2EBD	B	Assigns values to old (bit 1 FLAGX set) or new variables
tovr	2F64	B	Passes numbers from stack & strings from workspace to VARS area
let2	2F6D	B	Enters complete existing string as new string & reclaims old one
$tov	2F84	B	Transfers a newly declared sting to variables area
endv	2FA8	B	Adds a character to the end of VARS area and writes a new end byte (8Ø)
gstk	2FAF	B	Reads out the calc stack into BCDEA
dim_	2FCØ	B	Sets up space for new arrays in VARS, reclaims old ones if any
alnm	3Ø46	B	Returns C flag set if A hold digit or letter
alph	3Ø4B	B	Returns C flag set if A holds a letter
defp	3Ø59	B	Handles BIN and converts decimal nos. to fp form on calc stack
efor	3ØA9	B	Converts E-format entries to floating point on calc stack
nume	3ØD9	B	Returns NC if A holds a digit

stdg	30E0	B	If A holds a digit, that digit goes onto calc stack
stka	30E6	B	Puts absolute value in A onto calc stack (0-255)
stbc	30E9	B	Puts absolute value in BC on calc stack (0-65535)
infp	30F9	B	Puts line no. or integer in BASIC line on calc stack
dexp	310D	FF	Moves a general E-format decimal to calc stack
gint	313D	B	Gets a small integer (- to +65535) from (HL) into DE; sign in C reg
sint	314A	B	Stores small integer (- to +65535) at (HL) and next 4 bytes
fpbc	3160	FF	Compresses value on calc stack into BC, C set if too big, Z set if positive
alog	317F	FF	Gets log base 10 of 2 to power A into A
fpta	3193	FF	Gets number from calc stack to A; C set if overflow, Z set if positive
fppr	31A1	FF	Prints last value on calc stack to current print position
mtem	334A	FF	Executes A = 10 * A + C with carry returned in C
fadd	335A	FF	Prepares fp form for addition; complements negatives & replaces sign bit
fet2	3379	FF	Fetches 2 fp forms; first to H'B'C'CB, second to L'D'E'DE
shif	339C	FF	Shifts an fp form right to line up for addition
abak	33C3	FF	Adds back the carry when a number is shifted right
sub_	33CE	FF	Subtract routine; changes a sign and proceeds to add_
add_	33D3	FF	Floating point addition of two numbers
muli	3468	FF	Multiplies 16-bit integers: HL = HL * DE
fmul	347F	FF	Prepares fp form for mult or div; tests for 0, replaces sign bit
mult	3489	FF	FP multiplication; uses integer multiple for small integers
div_	356E	FF	FP division; exits via the mult routine
roun	35D3	FF	An fp op to truncate a number toward zero to integer form
texp	362B	FF	Tests exponent for large numbers; subroutine for roun
res2	3652	FF	Restacks two small integers in fp form
ress	3655	FF	Subroutine for res2, so the routine runs twice
rsta	3656	FF	FP op to send the number pointed to by HL to calc stack
cons	3684	F	File of constants in FP form: 0,1,.5,pi/2,10
fpfi	3696	F	File of addresses for FP ops. Use data display
fpop	371A	FF	Executes FP ops that follow RST 28. FP op interpreter
fpen	372B	FF	Re-entry point for the fpop routine
drop	3760	FF	Executes a return to drop a number from the calc stack
rafp	3761	FF	Takes contents of A and runs corresponding FP op for BASIC interpreter
tes5	3768	FF	Tests for 5 bytes more of memory for a new FP form
snum	3773	FF	Moves FP form to calc stack from elsewhere in memory
dup_	377F	FF	Duplicates a number on calc stack or moves a number to calc stack
stfp	3785	FF	Stacks fp form of a number supplied in code following op 34
stda	3787	FF	Gets data to calc stack as new FP number
szer	37B0	FF	Adds zeroes to calc stack to fill out FP form
fcon	37B6	FF	Finds needed constant in table of FP constants via A
badr	37C5	FF	Finds base address for each fp form in calc MEM area
get_	37CE	FF	Gets fp no. from calc MEM area to calc stac (get0 to get5)
stak	37DA	FF	Stacks one of the constants (0,1,.5,pi/2,10) according to 2nd nibble
stor	37EC	FF	Moves FP form from calc stack to MEM slot (stor0 to stor5)
swop	37FB	FF	Exchanges the order of last two FP forms on calc stack
sapp	3808	FF	Series approximator for calculating transcendentals (SIN, EXP, etc)
abs	3829	FF	FP op to make last calc stack value positive
neg_	382D	FF	FP op to change sign of last value on calc stack
nsin	3842	FF	Subroutine for ABS_ and NEG_ for small integers
sgnm	3851	FF	SGN op; returns 1 on calc stack for +, 0 for 0, -1 for -
inx_	3864	FF	Puts result of IN X onto calc stack

peek	386B	fp	Replaces last value on calc stack by contents of that memory address
usr#	3872	FF	Executes USR X, where X is last value on calc stack
rusr	3882	FF	Return routine for USR when cartridge is present
cusr	388E	FF	Checks for cartridge and if so sets up banks for USR call
nogo	38C5	FF	Way out of cusr if no cartridge present
gars	38CB	FF	Way out of cusr when cartridge is present (for USR)
usr$	38D7	FF	Executes USR$ from string parameters on calc stack
zert	39Ø4	FF	Tests FP form pointed to by HL for Ø, returns C set if so
n>Ø?	3914	FF	Tests last no. on calc stack & stacks 1 if positive, else Ø
not_	391C	FF	Executes NOT; stacks 1 if last value is Ø, else stacks 1
n<Ø?	3921	FF	Tests calc stack last value, stacks 1 if negative, else Ø
or_	3936	FF	Executes OR on two calc stack values
and_	393F	FF	Executes AND on last two calc stack values
$and	3948	FF	Executes AND between string (params on calc stack) and no. on calc stack
n=m?	395A	FF	Performs 12 <=> comparisons between Nos. and strings (from calc stack)
$tr+	39B7	FF	Executes string concatenation for two string params on calc stack
spnt	39DA	FF	Calc stack pointer set: HL to last value, DE to next
chr$	39E4	FF	Replaces X on calc stack by params of CHR$ (X)
val$	39F9	FF	Handles both VAL and VAL$, returns no. on calc stack
str$	3A3A	FF	Replaces X on _alc stack by params of STR$ X
rdin	3A6Ø	FF	Reads in character from channel (Ø-15) specified on calc stack
code	3A84	FF	Replaces params of A$ on calc stack by CODE A$
len_	3A8F	FF	Replaces params of A$ on calc stack with LEN A$
lonz	3A96	FF	Loop on non-zero (like DJNZ) using BREG as counter; FP op 35H
jru_	3AA1	FF	Jump relative unconditionally; followed by offset; FP op 33
jrt_	3AAA	FF	Jump relative on true on calc stack; FP op ØØH
end_	3AB6	FF	End an RST 28 calc and return to Z8Ø language
qrem	3ABB	FF	Replaces X and Y on calc stack by their quotient (last val) and remainder
int_	3ACA	FF	Replaces X on calc stack by INT X; 3ad2 continues FP code
exp_	3ADF	FF	Replaces X on calc stack by EXP X
ln_	3B2E	FF	Replaces X on calc stack by LN X
aadj	3B9E	FF	Reduces angle size for trig calcualtions; FF op 39
cos_	3BC5	FF	Replaces X on calc stack with COS X
sine	3BDØ	FF	Replaces X on calc stack with SIN X
tan_	3BF5	FF	Replaces X on calc stack with TAN X
atn_	3BFD	FF	Replaces X on calc stack with ATN X
asn_	3C4E	FF	Replaces X on calc stack with ASN X
acs_	3C5E	FF	Replaces X on calc stack with ACS X
sqrt	3C65	FF	Replaces X on calc stack with SQR X
pwr_	3C6C	FF	Raises last value on calc stack to power of next; continues at 3C78 as FP
tpfi	3C8A	F	File of ASCII cassette messages
asfi	3DØØ	F	ASCII character file (to end of ROM)

EXROM ADDRESSES

erro	ØØØ8	O	Error interrupt handler
xr38	ØØ38	O	Fields keyboard/clock interrupt when EXROM is in
xini	ØØ49	O	Initializer; enables all of home bank excpt chunk Ø
xout	ØØ4F	O	Disables and exits EXROM
boot	ØØ5A	O	Sets up xout at 6ØØØ as boot routine for BASIC ROM
svby	ØØ68	C	Subroutine to SAVE bytes to tape
rebo	ØØE5	C	Restores border color at end of a cassette op
ldby	ØØFC	C	Subroutine to LOAD bytes from tape
edge	Ø18D	C	Counts and times pulse edges during LOAD and VERIFY
cent	Ø1AB	C	Cassette op entry routine; op is in taddr; sorts for syntax
veri	Ø56F	C	Control routine for VERIFY
lblo	Ø5C6	C	Loads a block of bytes and returns
load	Ø5CC	C	Control routine for LOAD
merg	Ø6E5	C	Control routine for MERGE
melv	Ø7E8	C	MERGE a line or variable
save	Ø851	C	Control routine for SAVE
akey	Ø8AA	O	Waits for a keystroke
exin	Ø8E7	O	Initialization check for cartridge
lro?	Ø8FØ	O	Checks for presence of language cartridge and jumps to it
aro?	Ø9ØF	O	Checks for applications cartridge and jumps if there
lang	Ø91F	O	Tests for cartridge language
mlst	Ø928	O	Machine language start up for cartridge
sbas	Ø956	O	Starts BASIC applications cartridge
nova	Ø96C	O	Initializes SVs without leaving space for ml variables
bood	Ø99A	O	Boots highest priority device
bsct	Ø9F4	O	Builds current system configuration table
nram	ØADB	O	Test a new bank for RAM, moves in keyboard interrupt handler
asig	ØBD1	O	Assigns bank number to current bank
chir	ØC1F	O	Marks intelligent devices and initializes if initializable
cidi	ØC2F	O	Calls intelligent device initialization routine
rset	ØC4C	Ø	Performs RESET command on bus expansion unit
rnob	ØCFB	O	Renumbers expansion banks in order of interrupt priorities
opd2	ØDBØ	V	Opens DFILE2 and sets video mode
cld2	ØE27	V	Closes DFILE2 and clears video mode
svid	ØE6E	V	Switches video mode per value in VIDMOD
pass	ØF43	O	Fasses characters via bus expansion unit
jbnk	ØF8A	O	Jump interbank
cbnk	ØF99	O	Call a routine in another bank
xxxx	1ØØØ	O	ROM copy of RAM resident code; gets moved to 62ØØH
vtab	1DØØ	F	Table for fixing up addresses when RAM-res code is moved high
funf	1EDC	F	Jump table for functions in ROM; use data mode and ROM NAMEs
fun2	1FEC	F	Jump table for functions in EXROM; use data and EXROM NAMEs
fun1	1FD8	F	Jump table for RAM-res code; half wrong by one byte

HOT Z NAMES

$TIN	C2BØ	OP	Sends BC characters at (HL) to current screen position
2FIN	DB1F	DS	Searches file for a NAME at (HL)
3FIN	DB2Ø	DS	Searches file for a NAME at (DE)
4CHR	EØ41	OP	Sends a 4-charcter string to line print buffer
5BIN	C29F	OP	Sends 5-byte string at (HL) to current screen position
8AOP	E44D	AS	Codes for 8-bit arithmetic ops
ABRD	C276	OP	Reads from screen starting from column zero
ACKN	EØ9B	OP	Acknowledges valid keystrokes with beep
ACMD	EBC7	AS	Sorts assembly-edit commands
ACON	DA7C	DS	Prints (NNNN) forms in disassembly
ACSH	E3E5	AS	Codes for 16-bit ADD
ADDL	DFD3	OP	Performs HL = HL + A, preserves A
ADEN	C3C8	OP	Address entry point, test for NAME or hex
ADFN	E8A9	AS	Gets numeric address for a NAME
ADJR	DAA7	DS	Calculates destination address for JRs
ADNA	D3EB	SS	Prints address, three spaces, and corresponding NAME if any
ADVA	DE33	DS	Advance current address to next instruction address
ADVK	CEE1	ED	Advances edit cursor to the left
AFEX	BFDØ	WV	Single-Step value for AF'
AFRG	BFDC	WV	Single-step value for AF
AKIN	C2AD	OP	Sets 4 bytes at (HL) into top left corner
ALIN	D483	SS	Prints A register line in Single Step display
ALKE	C316	OP	Waits for a keypress, returns with key in A and C, B=ØØ
ALN2	BFEØ	WV	Third address slot for alternate NAME file parameters
ALNA	BFE2	4B	Write-in slot for alternate NAME file parameters, two addresses
ANNA	CA6B	NM	Gets ready for another NAME after rejecting one
ANYC	EØ1E	OP	Sends character in C to line print buffer B times
APRI	D4BA	SS	Prints A'register line for Single Step display
AR16	E3AE	AS	Codes for 16-bit ADC, SBC, ADD
ARE2	DF36	DS	Reads address at left of line in A
ARE3	D512	OP	Reads and address from screen and preserves BC
ASCD	C289	AS	Tests for ASCII hex digit (Ø-F), returns C set if not
ASED	EB2Ø	AS	Entry to assembly edit from READ mode (STOP command)
ASIM	5D3Ø	5B	Simulation area for single stepper, which runs ordinary steps here
ASRT	EACE	AS	Return point for assembly-edit commands
ATBC	E65F	AS	Codes for LD (BC),A
ATDE	E665	AS	Codes for LD (DE),A
ATOH	C37B	OP	Converts ASCII in A to hex
ATPO	C21E	OP	Sets screen print position corresponding to cursor attribute byte
AVCA	DEØ2	DS	Advance current disassembly address
B4SP	C2CD	OP	Backs print position 4 spaces for repeat address entry
BCEX	BFCE	WV	Single-Step value for BC'
BCRG	BFDA	WV	Single-Step value for BC
BERR	D9C7	DS	Prints ERROR after RST Ø8 and checks report number
BFCL	C32F	OP	Clears line buffer and prints disassembly screen
BFCO	BFB2	WV	Address of current position in line print buffer
BFKL	C252	OP	Kills contents of current line print buffer at 5DØ2
BIMN	E287	AS	Subroutine for assembly of BIT, RES, SET
BKAR	C2ØA	AS	Back arrow for assembly editor
BKSP	EBB7	AS	Backspace during assembly line edit
BKWD	DAB3	DS	Calculates destination address for backward JRs
BLAN	DAFF	DS	Prints blank if no NAME, else one space

BOIX	E2A6	AS	Assembles indexed bit ops
BORS	D66A	RC	BORDER color set command (BRIGHT)
BOUT	CØ63	CA	Break out routine from LD81
BPT1	BFBC	WV	First breakpoint address
BPT2	BFBE	WV	Second breakpoint address
BTOS	C4DC	OP	Sends line buffer to screen
CADR	BFFE	WV	Current address for disassembly
CAJP	E56D	AS	Subroutine for assembly of CALLs and JPs
CASC	C194	CA	Call to EXROM for 2Ø68 cassette routines
CASN	C79B	CA	Prompts for cassette name and puts it into cassette header buffer
CASO	C772	CA	Writes tape parameters to cassette buffer (5D8Ø)
CBDI	DC2D	DS	Disassemble bit ops (codes with CB prefix)
CBFI	FØ72	FI	File of mnemonics for CB instructions
CBFL	BFF5	BV	Byte flag for disassembly of CB instructions
CCOU	BFB9	BV	Delete this one
CDFI	F57E	JT	Command jump table (Step, Read, Edit, each starting with RND key)
CEOF	CD55	ED	Check whether END address is with 256 of cursor and if not ask for new value
CESC	EABF	AS	Escape from assembly when ';' key is pressed
CHA2	D9D5	DS	Converts hex value to ASCII and sends it to line print buffer
CHAR	C3Ø1	OP	Sends character in A to line buffer; preserves registers
CHGD	D764	DS	Changes display between Data and Disassembly
CHNA	CA65	NM	Sets values to change and existing label
CHOO	D774	DS	Selects Data/Disassemblv according to flag bit 4
CHPT	CAA7	NM	Entry point to WNAM when a NAME already exists for that address
CINS	EAED	AS	Inserts space at cursor during menmonics entry
CIR(	E9A7	AS	Checks mnemonic for initial (and returns Z or NZ
CIRA	E9A3	AS	Checks mnemonic for an initial A and returns Z or NZ
CIRS	E9AB	AS	Checks mnemonic for initial space and returns Z or NZ
CIRU	E9AD	AS	Sets 'initial' position, checks value against A and returns
CIT(	E9DD	AS	Check for 'initial' (in mnemonic and go to error trap if not
CITA	E9D9	AS	Check for 'initial' A in mnemonic and go to error trap if not
CITS	E9E1	AS	Check for 'initial' space and go to error trap if not
CITU	E9E3	AS	Set 'initial' position and compare with A, trap if not the same
CKIN	CDØB	ED	Checks insert flag and excutes insertion
CKR(	E997	AS	Checks for (and returns Z or NZ
CKRA	E99B	AS	Checks for an A and returns Z or NZ
CKRH	E993	AS	Checks for an H and returns Z or NZ
CKRS	E99F	AS	Checks for a space and returns Z or NZ
CKRU	E9B5	AS	Advances position counter, checks value against A, returns
CKRX	E98F	AS	Checks for an X and returns with Z or NZ
CKSS	C161	AS	Gets top line of active screen for assembly or single step screens
CKT(	E9CD	AS	Check for (in mnemonic and go to error trap if not
CKT)	E9C1	AS	Check for) in mnemonic and go to error trap if not
CKT+	E9C5	AS	Check for + in mnemonic and go to error trap if not
CKTA	E9D1	AS	Check for A in mnemonic and go to error trap if not
CKTI	E9BD	AS	Check for an I in mnemonic and go to error trap if not
CKTL	E9B9	AS	Check for an L in mnemonic and go to error trap if not
CKTS	E9D5	AS	Check for space in mnemonic and go to error trap if not
CKTU	E9EB	AS	Compare character in mnemonic with A, trap if not the same
CKTV	E9C9	AS	Check for a comma in mnemonic and go to error trap if not
CLEN	C2CØ	OP	Clears an invalid NAME from screen

CLLI	EØ18	OP	Fills line print buffer with 32 spaces (2ØH)
CLMM	CBB4	EC	ERASE command handler, fills cursor to END with ØØ
CLMN	DE1F	DS	Clears old mnemonic from display screen prior to printing current one
CLOS	C5A9	NM	Closes gap in NAME file after a move
CLWA	EØ67	OF	Clears BASIC's work area to remove old address entries
CMPO	E8DE	AS	Determines position of comma in a mnemonic entry
CNAM	C73A	CA	Gets a tape name for cassette ops
CNBA	EE7B	FI	File of ASCII conditional particles
COCT	DDF9	DS	Gets second octal digit of A into A
CODE	DE3A	DS	Get instruction length and print hexcode column
COFP	D97B	DS	Interprets f-p constant-to-stack operators
COLR	EBDA	RC	Gets in color number for INK, PAPER, BORDER commands
COMP	C2FØ	OP	Prints comma to line buffer
CONL	DA58	DS	Disassembles conditional forms, Z, NZ, etc.
CORN	E25B	AS	Gets (C) or (NN) for assembly of INs and OUTs
COUN	BFB6	WV	Pointer for printing register display; points to register names
CPAR	C2F8	OF	Prints closing parens to line buffer
CPBC	DB8A	DS	Compares BC and DE, returns Z for match, NC if DE larger
CPFI	EFC5	FI	File of conditionals for disassembler
CPFI	F297	FI	Conditional particle file for disassembler
CREG	DC8E	DS	Identifies first register in 8-bit LDs
CRST	D261	SS	Handles RUN CALL command for RSTs
CRUN	D714	SS	Loads all registers, runs step, saves all registers
CSBF	5D8Ø	BF	Buffer for cassette tape header; use data mode
CSUM	EDØ7	EC	Checksum command (LEN)
CTSC	EAFE	AS	Checks for space or comma; used after conditionals
DADR	DADØ	DS	Prints 16-bit number or address NAME for disassembly
DAFI	F54E	JT	Disassembler mnemonics argument jump table
DATL	D787	DT	Prints one line of data display
DATP	D77E	DT	Prints full screen of data display
DBEN	E121	AS	Puts DB bytes into memory and redoes screen to hide them
DBL1	EE74	FI	File of second character of double register names
DBLE	C11E	EC	Resets stack when hexedit cursor is called from hexedit
DBLF	EFBB	FI	File of double register names
DBLR	F28A	FI	Double register file for arithmetic ops
DCIN	ED3A	DS	Gets in decimal address for next disassembly page
DCKS	CF84	ED	Redoes Data display after backing up one address
DDAT	D7AD	DT	Main routine for printing data display
DDLD	E6B7	AS	Codes for LD RR,NNNN (direct double load)
DEEX	BFCC	WV	Single-Step value for DE'
DELE	EC42	AS	Removes a character from screen during assembly edit
DENA	CAC6	NM	Delete-NAME command handler (EXP)
DERG	BFD8	WV	Single-Step value for DE
DEWD	DFDF	DS	Sends hex number in DE to line buffer for printing
DHED	EØ4D	DS	Prints disassembly screen column headings
DIRL	E5FF	AS	Codes for direct index register LD
DISØ	DB98	DS	Disassemble op codes from ØØ to 3F
DIS1	DC78	DS	Disassemble op codes from 4Ø to 7F (8-bit LDs)
DIS2	DCØD	DS	Disassemble op codes from 8Ø to BF
DIS3	DB9D	DS	Disassemble op codes from CØ to FF
DISA	D759	DS	Main disassembler loop

DISP	DDD1	DS	Sorts direct loads to IX/IY from indexed displacements
DISS	DBAØ	DS	Main disassembly loop
DIVI	D7E1	DT	Divides HL by BC for decimal conversions
DLHL	E656	AS	Codes for LD (HL),N
DLIS	C9F4	EC	Sends lines to 2Ø4Ø printer from cursor to END (LLIST)
DON?	DF8E	NM	Checks whether a NAME look up is completed
DPAG	DAØB	DS	Disassembles and lists to end of screen
DSCO	DEDD	DS	READ mode command point, waiting for entry
DSKR	C939	AS	Gets top address on screen, sets print parameters for a down scroll
DSPA	EØ25	OP	Prints double space
DSWI	DF9E	RC	Data/disassembly display switch, THEN command in READ
DTFI	EFD2	FI	Various disassembler text messages
DUM2	D737	UU	Entry point for DUMP utility
DUMP	CF4Ø	UU	Dumps all register values to Single Step; a users' utility
E2FI	EFE3	FI	Disassembler mnemonics for low ED instructions
EADR	DF1C	DS	Reads entered address from ADDR slot at top left
EBAK	CEBA	ED	Backs blink bit for cursor left, escapes if too far
ECMD	CDAA	ED	Calculates offset into jump table for EDIT commands
ED1I	DCD3	DS	Disassembles op codes ED4Ø to ED7F
ED3I	DD6B	DS	Disassembles op codes from ED8Ø to EDBF
EDAT	CE43	ED	Data mode edit routine
EDBK	CE88	ED	Backs cursor during edit
EDCO	CDBB	ED	EDIT command point, waiting for key entry
EDDI	DD3E	DS	Sorts ED prefixed ops for disassembly
EDES	CE7F	ED	Escapes from the middle of an edit entry via ENTER
EDFI	EF8Ø	FI	Mnemonics file for disassembly of high ED instructions
EDIT	DØ4D	ED	Sets up cursor at first hexedit position
EDMD	CDB5	RC	Turns on EDIT mode, changes headings, sets cursor
EDRT	CDB8	ED	EDIT command return address
EERT	EBB1	AS	Address on stack used by syntax error trap
ENCN	C3BD	OP	Loop for entry of characters of a NAME or address
ENDA	BFEA	WV	Current address in END
ENDE	CDD7	ED	Ends a line edit, moves down cursor, reenters loop
ENNA	DF58	DS	Entry loop for NAME at top left
ENTN	C3B7	OP	Entry point for NAME entry
ENTP	C3B4	OP	Entry point for END, DEST, LOOK at top left
ENTR	DFØB	DS	Looks up address/NAME entries
ERAS	EBØ7	AS	Deletes character behind cursor during mnemonics entry
EREN	EBAA	AS	Re-entry point after error trap
EROP	EAB1	AS	Continues syntax error processing
ETRI	E315	AS	Local assembly error trap
ETRJ	E244	AS	Local assembly error trap
ETRK	E171	AS	Assembly local error trap
EVAD	E8C1	AS	Evaluates address (ADDR) for assembly
EXFL	D4C4	SS	Prints exchange flag value to Single Step screen
EXFA	C462	AS	Decodes mnemonics of EX (SP) instructions
FCBQ	BFF3	BV	Holds displacement byte for indexed bit operations
FDAR	C211	AS	Forward arrow for assembly editor
FDSP	EB01	AS	Right arrow during assembly line edit
FENS	BF9E	FI	Switch address for Single Step window/no-window; starts initialization file
FIØ8	E925	AS	Gets a number from Ø to 7 or reports error

FIØF	E931	AS	Tests ASCII for hex digit ØØ to ØF and converts to hex
FIAG	C946	EC	Find again command (ABS) to locate subsequent matches to string
FIAT	C396	OP	Find attribute address for current print position; set DE to input buffer
FILC	BFB4	BV	ASCII value of fill character for Fill command
FILI	CCØ2	ED	Finds current edit line
FILF	CBFC	OP	Fills screen after an editing operation
FIMN	E9F2	AS	Finds match in table for the first 3 letters of the mnemonic
FINA	DB3C	DS	NAME file search routine, expects address in DE else enter at F2NA
FIND	DB1C	DS	Searches NAME file for a NAME at CADR
FINI	D74A	SS	Entry to CRUN for some simulation routines
FINS	CF12	ED	Tests first instruction for type and length
FIXD	EA3C	AS	Assembles invariant mnemonics, e.g., CCF, SCF
FLAG	D4CC	SS	Prints flag values to Single Step screen
FLCK	D698	DS	Checks disassembly flags for RST 28 or RST Ø8 in progress
FLMM	C6A1	EC	Fill memory command handler (FN)
FLOT	D8AD	DS	Begins f-p interpreter for RST 28
FPAG	DØC7	DS	Finishes a disassembly screen to bottom
FPAT	ECC1	DS	Continues floating point disassembly
FFCF	F334	FI	File of floating-point constants (ASCII)
FPDA	D958	DS	Main floating-point data interpreter
FPFI	F43Ø	FI	File of floating point mnemonics
FPFL	CC28	DS	Sets flags indicating floating-point disassembly in progress
FPJR	D9AB	DS	Interprets f-p op relative jumps
FPSW	DØD7	RC	Toggles f-p interpreter mode (CODE)
FREG	DC89	DS	Adds final register to 8-bit register LDs
FTOB	D9AØ	DS	Moves characters from mnemonics file to line buffer
FWCJ	E49Ø	AS	Codes for conditional JRs entered with +N displacement
FWDJ	E48E	AS	Codes for JRs entered with +N for displacement
FXBA	EE6Ø	FI	File of codes of 'fixed' mnemonics
GALF	D233	ED	Gets in an alphanumeric character, rejects others
GBAK	D3Ø1	SS	Handles EDIT key to back up one step or byte
GDEC	C8BØ	RC	Gets in decimal address in READ mode
GDSP	C614	ED	Gets in displacement for readdressing commands
GHDG	D22A	ED	Gets in a hex digit, rejects other characters
GLIN	C488	OP	Selects a single line of data or disassembly to fill screen
GNGO	CCF9	ED	Go/no-go routine for Transfer, Find, etc.
GOBP	CØ96	SC	Main run-to-breakpont routine; saves window setting, forces no window
GOMO	C872	SS	Continues run-to-breakpoint, gets or skips window screen
GOSV	D245	SS	Handles value entries on Single Step screen
GOTI	CE2B	ED	Installs code into memory after EDIT entry
GQUO	E7B5	AS	Checks quotes and gets character into E
GTOF	CBD3	OP	Gets top of current working screen & renews to bottom
GTOS	C15E	AS	Finds top of active screen area for assembly or single-step screen
GUPA	DØ15	ED	Gets address at line 22 for screen-up
HARI	C8Ø4	EC	Hex arithmetic command; prints END + cursor addr, END - Cursor addr
HBYT	DFE4	DS	Sends hex byte in A to line buffer for printing
HEAD	DAE4	DS	Prints READ mode column headings
HED1	EECC	FI	Characters for disassembly column headings
HED2	EEAC	FI	Characters for data display column headings
HED3	EE8C	FI	Characters for single step column headings
HELO	CC64	OP	Gives beep on start up or error return

HIDG	E91C	AS	Shifts high nibble of hex byte left
HLEX	BFCA	WV	Single-Step value for HL'; storage from step to step
HLIX	DABB	DS	Distinguishes HL, IX, and IY in disassembly
HLRG	BFD6	WV	Single-Step value for HL
HOLD	D2D7	SS	Wait point for Single Step command entry
HTOA	EØØ7	OF	Converts hex to ASCII in A
HUNT	C977	ED	Main search routine for FIND
HWOR	DFDB	DS	Sends current disassembly address to line buffer; address column
HZET	ECE7	OF	Fields the ROM error traps when HOT Z is running
HZFG	5C73	BV	Flag byte with 5C73 to control HZ modes; see notes
IADA	C91C	AS	Calculates addresses and moves NAMEs for an insert
ID16	E378	AS	Identifies 16-bit register pair for INC/DEC instructions
ID81	E371	AS	Codes for INC/DEC (IX/Y+NN)
IDIX	E393	AS	Codes for INC/DEC IX/Y
IDLD	E6D4	AS	Codes for LD RR,(ADDR) (indirect double load)
IESC	CCDB	ED	Escape from insert when instruction will not fit
IFCH	EØ3B	OF	Filters non-print characters before sending to line print buffer
ILEN	DE62	DS	Look up length for instruction byte at (HL)
IMAR	C1F4	SS	Sets cursor for single-step register value entry
INAJ	C8EØ	AS	Saves various registers while calling routine to set up insert addresses
INBY	CØ1Ø	CA	Gets in one byte from ZX tape
INCK	CDDC	ED	Checks for insert key (EDIT)
INCO	EA6A	AS	Locates entry point for code entry, handles insertions
INDO	EA8E	AS	Resets disassembly after assembled code is entered
INDX	DDC8	DS	Gets displacement for indexed forms, prints, closes parens
ININ	CEC4	ED	Gets in hex code instruction to screen
INKS	C426	RC	INK color change command handler
INRE	E73D	AS	Codes for LDs to and from I and R registers
INSS	C8C6	AS	Finds line if no insert, else jumps to insert routine, for code entry
INSY	DDA4	DS	Gets system variable NAMEs for indexed displacement reference
INYE	DD9C	DS	Prints indexed displacement from IY for unNAMEd slots among SVs
IOFF	DØ37	ED	Switches off insert flag when cursor is moved
IORG	E247	AS	Gets I/O register for IN X,(C) or OUT (C),X
IRDR	E1B5	AS	Subroutine for assembly of LDI, LDIR, LDDR, LDD and similar instructions
IRED	C272	ED	Reads Z8Ø instruction from hex digits on screen
IVAR	F3CE	FI	File of initial HOT Z system variables for startup
IX+N	E88A	AS	Checks and codes for (IX+NN) forms
IXIY	DBDD	DS	Set FDDQ flag for DD or FD prefixes
IXRG	BFD4	WV	Single-Step value for IX
IYRG	BFD2	WV	Single-Step value for IY
JCMD	DF89	EC	Sets table base for edit command jump table
JPHL	DFCA	OP	Jumps to Ath address in jump table at (HL)
JRDI	E4F4	AS	Calculates displacement for relative jumps
JUST	D8Ø2	DT	Right justifies decimal numbers
K-ON	DØ51	ED	Turns on a top-line cursor at left
K-UP	CF6C	ED	Handles cursor up commands
KADD	BFEE	WV	Address on screen next to cursor, from ARED or KRED
KATT	BFAE	WV	Address of screen-cursor attribute, for setting blink or bright
KBRI	C218	OF	Sets bright cursor
KDWN	CF57	ED	Cursor down routine
KEYB	EØ6E	OF	Keyboard read; waits for a key, beeps, controls modes

KHED	EØ52	DT	Prints data screen column headings
KLIN	BFF1	BV	Screen line number of line with cursor, for cursor controls
KLOC	DØ54	ED	Records cursor-line as top scren line
KLOD	C76Ø	OF	Loads character in A into a cursor
KMVS	CDCA	ED	Handles cursor moves during EDIT
KOUT	DØ31	ED	Turns out cursor
KPOS	BFFØ	BV	Print position on screen for cursor
KRED	DF33	OF	Reads address at left of cursor line
KRES	DØ48	ED	Restores cursor at former position after a command
KRGT	CDE5	ED	Checks for cursor right, then DELETE, then command keys
KSRT	EBA9	AS	Moves line cursor right
KURS	DØ5D	ED	Records cursor attribute byte and sets blink/bright
LD68	C731	EC	2Ø68 LOAD command handler
LD81	CØØØ	EC	Loads ZX tapes to addresses from cursor to END
LDAD	E7F7	AS	Codes for an address when assembling LDs
LDIN	DDDE	DS	Handles direct loads to IX/IY
LENI	BFE6	BV	Length of current instruction during assembly, in bytes
LFPØ	BFBØ	WV	Address of last floating-point disassembly line for f-p interpreter
LFPD	D91F	DS	Lists floating point data as decimal
LINE	EØ1Ø	OF	Sets screen position to BC and draws 32 character line across
LNAM	D87E	DT	Print NAME column for Data display
LNFI	EEEC	FI	File of instruction lengths
LODN	C7F5	EC	Gets in tape name for a 2Ø68 LOAD
LOOK	DBE9	DS	Sorts instructions for disassembly look up
LOSI	BFC4	WV	Last Single-Step instruction address; top line of disassembly
LTDF	C1FA	OF	Converts screen line number to display file address in HL
LURP	DA86	DS	Look up register pair for disassembly
M-CP	E464	AS	Assembles CP instructions
M-DB	EØE9	AS	Assembles DB when used with hex numbers (no quotes)
M-EX	E174	AS	Assembles EX instructions
M-IM	E153	AS	Assembles IM instructions
M-IN	E224	AS	Assembles IN instructions
M-OR	E46Ø	AS	Assembles OR instructions
M-RL	E2C6	AS	Assembles RL instructions
M-RR	E2CA	AS	Assembles RR instructions
MADC	E43B	AS	Assembles ADC instructions
MADD	E437	AS	Assembles ADD instructions
MALD	E5B2	AS	Assembles LD instructions; sorts on comma position
MAND	E42C	AS	Assembles AND instructions
MAT?	C96F	ED	Tests for match with search string
MATS	C954	EC	FIND command handler (SGN)
MBIT	E27D	AS	Assembles BIT instructions
MCAL	E56Ø	AS	Assembles CALL instructions
MCND	E93D	AS	Subroutine for assembly of conditional mnemonics
MCPD	E1D9	AS	Assembles CPD, CPDR
MCPI	E1CD	AS	Assembles CPI, CPIR
MDAO	E47Ø	AS	Codes for direct arithmetic ops (e.g., ADD A,NN)
MDB'	EØD2	AS	Handles assembly op DB when used with quoted string
MDEC	E352	AS	Assemble DEC instructions
MDJN	E4E7	AS	Assembles DJNZ
MFIN	DEØA	DS	Finds numbered entry in A in table at (HL), end bits 7 set

MFOU	C9BØ	ED	Match-found escape from HUNT; displays matching location
MHAL	E149	AS	Assembles HALT instruction
MINC	E34E	AS	Assemble INC commands
MIND	E1DD	AS	Assembles IND, INDR
MINI	E1D1	AS	Assembles INI, INIR
MJPC	E52C	AS	Assembles JP instructions
MJRS	E4AE	AS	Assembles JR instructions
ML23	E7CF	AS	Codes for 8-bit register to register loads, LD R,R'
ML24	E7C1	AS	Codes for direct 8-bit register loads, LD R,NN
ML25	E7AA	AS	Codes for LD R,'A', where A is ASCII character
ML26	E773	AS	Codes for LD R,(RR), where RR is HL, IX/Y, DE, BC
ML28	E766	AS	Codes for LD A,(ADDR)
ML29	E759	AS	Codes for LD R,(IX+NN), where R is an 8-bit register
MLD2	E7Ø8	AS	Codes for LD R,X, where X is any option
MLD3	E679	AS	Codes for LD RR,XX, where XX is direct or indirect expression or HL
MLD5	E61C	AS	Codes for LD (RR),X
MLD7	E6ØB	AS	Codes for LD (ADDR),RR
MLD8	E5D9	AS	Codes for LD (IY+NN),XX,where XX is register or number
MLDD	E1D5	AS	Assembles LDD, LDDR
MLDI	E1D9	AS	Assembles LDI, LDIR
MLIN	CEAE	OP	Remakes one line when new instruction is same size as old
MNAD	F4F6	JT	Assembler routine jump table
MNAM	DAD8	DS	Looks up NAME and prints it or address if none
MNAR	DCC7	DS	Takes argument from mnemonics file and jumps to handler routine
MNBA	EDAD	FI	File of mnemonics for assembly
MNEM	DDØ5	DS	Reads mnemonics from file to line buffer, traps argument byte
MNFI	FØ9C	FI	File of mnemonics for main instruction sequence
MNLO	EB7A	AS	Mnemonics edit loop for entry
MNPR	DD87	DS	Finds mnemonic in file and sends it to (DE)
MNUP	C5CE	NM	Moves part of NAME file up to reorder NAMEs
MOTD	E1E5	AS	Assembles OTDR
MOTI	E1E1	AS	Assembles OTIR
MOUT	E1EE	AS	Assembles OUT, OUTI, OUTD instructions
MOVE	CB6B	ED	Subroutine for transfers, moves code in proper direction
MPOP	E318	AS	Assembles POP instructions
MPUS	E31F	AS	Assembles PUSH instructions
MREG	D410	SS	Prints main registers and their current values
MRES	E281	AS	Assembles RES instructions
MRET	E2E2	AS	Assembles RET instructions
MRLC	E2AF	AS	Assembles RLC instructions
MRRC	E2B5	AS	Assembles RRC instructions
MRST	E1A4	AS	Assembles RST instructions
MSBC	E43F	AS	Assembles SBC instructions
MSET	E285	AS	Assembles SET instructions
MSLA	E2D3	AS	Assembles SLA instructions
MSRA	E2D7	AS	Assembles SRA instructions
MSRL	E2DB	AS	Assembles SRL instructions
MSUB	E42B	AS	Assembles SUB instructions
MVNA	C5E8	NM	Computes addresses for moving NAMEs
MVNM	C576	NM	Suroutine to move a NAME
MXOR	E43Ø	AS	Assembles XOR instructions

NACK	E8F5	AS	Checks whether a sequence of characters is a NAME
NADD	BFFC	WV	Next address for disassembly
NAME	DB08	DS	Looks up NAME at CADR and prints if there is one
NAMV	C564	NM	Main routine for moving NAMEs
NAPA	C243	DS	Prints NAMEs in column 14 of disassembly
NARO	C3DE	NM	Erase/backspace handler for NAME entry
NASW	BFF8	WV	NAME file switch address; off if = NEND, on if = NTOP
NCOL	DAF9	DS	Prints disassembly mode NAME column
NCUR	C3AE	NM	Set NAME cursor
NENT	C83B	EC	Gets in a NAME in data mode, jumps if disassembly NAME assignment
NESC	C369	NM	Escape routine during NAME entry (when ENTER is hit)
NEWK	CF62	ED	Calculates new cursor position from HL + DE and sets it
NFOU	C9AB	ED	Not-found escape from HUNT
NOBA	C325	OP	Searches stack for BASICs return address and jumps to it
NORA	E7FA	AS	Assembler check routien for NAME or address
NOSI	BFC6	WV	Next Single-Step instruction address
NSGN	D7ED	DT	Handles alignment of minus sign on negative decimals
NTDN	CB09	NM	Moves NTOP down for a NAME to be added
NTOP	BFF6	WV	Address of low expanding edge of NAME file
NTUP	CB04	NM	Moves NTOP up after a NAME deletion
NUMB	DA65	DS	Gets 8-bit hex digit to disassembly for direct loads, etc.
NWRV	D1F6	SS	Installs new register value in register display
OCEX	DDF0	DS	Exchanges 2nd and 3rd octal digits of a hex byte
OHED	E048	SS	Prints Single-Step screen column headings
OKIN	EBAE	AS	Mnemonic is ready; put it in
OKLO	D056	ED	Records cursor line from A and attribute byte from DE
OLIN	D9ED	DS	Disassembles a single instruction and prints line
ONES	D53A	SS	Reads EDIT cursor and runs the instruction there
OPAR	C2F4	OP	Prints open paren to line buffer
OPES	EC5A	AS	Opens space in assembly edit line for insertion
OSAS	D29E	SC	Enables assembler loop from Single Step (STOP)
OSBS	E0B2	SS	Handles value setting for A and F registers in step mode
OSCM	D33B	SS	Processes Single Step commands
OSCO	D2D0	EC	Enters Single Step from EDIT (STEP) and runs step at cursor
OSDF	BFC2	WV	Address of Single-Step window's display file
OSDP	BFC0	WV	Address of Single-Step display point in window
OSEN	DF6C	SS	Entry loop for NAME at line 18 in Single Step
OSNA	D31F	SS	Handles NAME entry to Single Step screen
OSOU	CDA3	SS	Single Step exit, returns to READ mode
OSRS	D193	SC	Set register values command handler (VAL)
OSRT	D771	SS	Single-Step return point to READ
OVER	BF70	LB	Label marker for top of user single step stack
PAPS	C412	RC	PAPER color change command handler
PBOT	D96F	DS	Records last floating point operation in LFPO at bottom of screen
PCH$	D7C4	DT	Looks up BASIC tokens and prints them
PCON	DAC7	DS	Prints forms (HL), (IX), (IY) for disassembly
PDAD	D865	DT	Print decimal address column for data display
PDAT	DD4C	DS	Prints DATA for invalid ED instructions
PDIS	C1E6	DS	Prints disassembly to screen bottom, restores cursor
PEOP	CD7C	ED	Prints END value on screen when EDIT cursor is on
PERR	DD61	DS	Prints ERROR for RST 08 instruction sequel

PEXP	D51F	SS	Prints 'EXFLAGS'
PFIL	D08A	DS	Prints from any file with bit 7 set for last character
PFLA	D524	SS	Prints 'FLAGS'
PFPC	D8C1	DS	Prints floating point column in disassembly
PFPO	D9AØ	DS	Looks up and prints floating point operator mnemonics
PHLT	DD51	DS	Prints HALT mnemonic
PINS	EØC3	ED	Gets length of instruction at insert cursor
PL-N	EC15	SS	Prints LAST-NEXT on register display
PLAD	D8ØF	DT	Prints last digit of decimal number
PNIB	DFF1	DS	Sends one nibble of hex byte to line buffer for printing
POIN	BFBA	WV	Single Step's pointer for reading register values
POKI	CE93	ED	Inserts code into memory at proper address
PPAG	C23B	DS	Sets up to print screen from last cursor address
PPIX	E33F	AS	Subroutine for PUSH/POP of IX/IY
PPIY	D111	SS	Subroutine for simulation of POP/PUSH IX/IY
PRAT	C38D	OP	Set print parameters at line and column in BC
PRI$	EØ2A	OP	Sends character string of length BC at (HL) to line print buffer
PRIM	BFB6	BV	Holds Ø or 27H for registers or exchange registers; printed
PRLD	DD59	DS	Prints LD
PRNA	C2FC	OP	Prints 'A' to line buffer
PRSC	CA5F	RC	Print-screen command for Read and Single Step
PRWS	C1EF	EC	Does a print-screen to 2Ø4Ø in EDIT mode
PSCR	C1B9	EC	Part-screen command; gets address and disassembles to bottom of screen
PSDB	D846	DT	Print 8-bit signed decimal (-128 to 127)
PSDW	D83B	DT	Print 16-bit signed decimal
PSSP	D52C	SS	Prints 'SP '
PSTA	ECØD	SS	Prints STACK on register display
PTOP	D76C	DS	Reprints display from top (line 2)
PUDB	D84F	DT	Print 8-bit unsigned decimal (Ø to 255)
PUDN	CD39	ED	Pushes down memory contents to make room for insert
PUDS	D518	DT	Prints unsigned decimal byte and a space
PUDW	D84Ø	DT	Print 16-bit unsigned decimal
RADD	C62D	EC	Command handler to readdress a jump table (STR$)
RANA	C3FØ	EC	Reassigns NAMEs to a diplaced area of memory (CHR$ command)
RCAL	D271	SC	RUN CALL command handler (INT)
RCMD	DEF5	DS	Sets return address for READ commands, looks up and jumps to command
RDBL	E87Ø	AS	Identifies 16-bit register pair for coding
RDHX	C346	OP	Reads a hex digit from the screen at BC, returns it in A
RDIS	D348	SS	Prints Single Step screen
RDIT	EA57	AS	Main routine for reading back and assembling mnemonics
RDRS	D9BB	DS	Return point for RST 28, Ø8 disassembly routines
RDUP	EC74	AS	Reads mnemonic entry from screen to buffer at 5D15 for syntax check
REAC	DECA	DS	Reactivate ADDR cursor after invalid NAME entry
REDO	CC15	ED	Redoes the screen after edit operations
REG8	E97B	AS	Looks up 8-bit registers in table
REIN	E7E3	AS	Codes for LD A,I and LD A,R
RELO	C65Ø	EC	Command handler for relocator (MOVE)
RESC	D39F	SS	Resets lower part of screen only
RESK	C2A4	OP	Resets the address cursor at top left
RETE	EAA7	AS	Return point for syntax error traps, flags errant character
RG16	E961	AS	Looks up 16-bit registers in table

RG8F	EF6C	FI	File of 8-bit register names
RGX8	E842	AS	Identifies 8-bit registers for coding
RGXF	E83E	AS	Checks syntax and gets displacement for IX+NN forms
RHEX	E9ØF	AS	Reads a hex byte from mnemonic to E
RIA2	DF24	DS	Reads a hex address at left of line in A
RIAD	DF23	DS	Reads a hex address from screen at top left
ROIN	DC69	DS	Disassemble rotate and shift instructions
RSFP	D91A	DS	Reads floating point data from code stream
RSPA	C358	OP	Moves print position 1 space right with wrap around
RSTD	DA26	DS	Disassembles RST instructions
RTBP	DØED	SS	Runs steps and checks whether a breakpoint has been reached
RUNT	CA37	EC	RUN command handler; transfers control to code at cursor
SAVN	C7FA	EC	Gets in tape name for a 2Ø68 SAVE
SBP1	D14D	SC	Set breakpoint 1 (AT)
SBP2	D159	SC	Set breakpoint 2 (OR)
SCDE	DFFA	OP	Sends hex number in DE direct to screen
SCDN	CF9E	ED	Scrolls screen down and finds an instruction to fill the line
SCND	DE16	DS	Tests screen bottom, returns NC if last line printed
SCPP	C172	AS	Gets scroll line for assembly or single step screen
SCUP	CFE5	ED	Moves screen up for cursor at bottom
SDBY	D854	DT	Convert to signed 8-bit decimal
SDFC	C2D8	OP	Sets DF_CC from current S_POSN, returns S_POSN in HL
SDON	D733	SS	Entry to CRUN for some simulation routines
SDRG	D1EC	SS	Sets new value for SP (USRS) in register display
SDWO	D817	DT	Convert to 16-bit signed decimal
SEND	EC36	RC	Sets END from READ mode (TO command)
SEOP	CD66	ED	Handles TO command to set END
SETF	D165	SS	Sets flags register values in register display
SFLA	D1BF	SS	Gets in new setting for flags register
SHBP	D122	SS	Displays current breakpoints (AND)
SHLP	DB92	OP	Set HL' to proper value for return to ROM
SHWT	C9EC	ED	Displays new screen starting at HL
SICA	D649	SS	Simulation routine for stepping CALLs
SIFI	F562	JT	Single Step simulation jump table
SIJP	D62D	SS	Simulation routine for stepping JPs
SIJR	D6D8	SS	Simulation routine for stepping JRs
SINC	D618	SS	Simulation routine for stepping INC/DEC SP
SINS	CBFØ	ED	Sets insert flag and checks for valid END
SJPH	D6B4	SS	Simulation routine for stepping JP(HL/IX/IY)
SkID	D9DB	DS	Skips over ordinary disassembly for RST Ø8, 28 ops
SkIP	D2E9	SS	Handles space key to skip one step
SkRL	CFCC	RC	Handles scroll (<>) command; scrolls until BREAK
SkUR	DØ6Ø	ED	Sets cursor blink and bright if caps shift untoggled
SOFF	EC1E	RC	Sign off; installs current NAME file as permanent, goes to BASIC
SORC	EC99	AS	Searches mnemonic string for first space or comma
SORT	D57Ø	SS	Sorts for simulation type of step instruction
SPAC	C2EC	OP	Prints space to line buffer
SPAP	E8F1	AS	Determines next blank space position in a mnemonic entry
SPBI	BFDE	WV	Storage bin for stack pointer during Single Step
SPON	DØCE	RC	Toggles flag to enable or disable SP display (AT)
SPPO	C37Ø	OP	Sets current position in line buffer to value in C

SPRD	DØØ1	RC	Reads machine stack pointer and prints it upper right
SPUP	D67Ø	SS	Simulation routine for stepping PUSH/POP
SRET	D67F	SS	Simulation routine for stepping RETs
SRST	D6ØØ	SS	Simulation routine for stepping RSTs
SSOR	D55E	SS	Sorts step instruction, selects simulation routine if needed
SSPH	D627	SS	Simulation routine for stepping LD SP,HL
SSPL	D69B	SS	Simulation routine for stepping LD SP,NNNN
SSPO	C381	OP	Set screen print position from line and column in BC
SSPT	D6A9	SS	Simulation routine for stepping LD SP,(NNNN)
SSWA	C73E	SS	Code to be copied into single step work area for code simulation
STAK	D3CC	SS	Prints current user's stack on single step screen
STAR	C4FØ	OP	Initialize and start up HOT Z
STE2	D54Ø	SS	Steps current instruction in NOSI
STEN	EBA5	AS	Start entry by printing initial character to screen
STEP	D53D	SS	Sets up simulation area and runs current instruction as a step
STTL	DØC1	OP	Clears line buffer and sets screen position to top left
STWD	C89Ø	SS	Stops window if flag set; restores screen after window
SUTR	CB4B	ED	Sets up transfer parameters, gets DEST
SUWA	D7Ø4	SS	Sets up stepper work area in printer buffer
SV68	C721	EC	2Ø68 SAVE command handler
SVAR	D24F	ED	Handles value entries at top left of screen
SWAS	EA1C	EC	Switch from hexedit to assembly edit (STOP command)
SWDD	CB17	EC	Switches disassembly/data displays during EDIT (THEN)
SWFP	DØ26	RC	Switch floating point interpreter; PEEK command
SWIN	DØA3	SC	Checks if there has been a window, switches it IN if so
SWNA	CA43	NM	NAME switch (OVER command) to change label files
SWOU	C1BØ	SC	Switch out window; single step OUT command
SWPM	C26Ø	ED	Sets up parameters for entry in EDIT
SWTE	CC4A	EC	Switch-to-edit command (>=)
TEM1	5C9E	WV	Description: First of 9 temporary word storage bins, mostly for relocations
TEM9	5CAE	WV	Last local word variable storage bin
TEND	CØ88	CA	Tape-end check routine for LD81
TERM	D314	SS	Exits from Single Step to READ mode
TIXY	E88D	AS	Checks and codes for IX+NN)
TLSC	C299	OP	Sets print position for top left of screen
TOPK	C312	OP	Sets main ADDR cursor at top left
TOPN	C9E5	RC	Displays beginning of NAME list (RND command)
TRAN	C537	EC	TRANSFER command, copies memory contents to DEST
TREG	DC97	DS	Identifies second register in LDs or register in arithmetic ops
TRNA	C53D	EC	Copies memory and moves NAMEs to DEST; MERGE command in EDIT
TXFI	F346	FI	Various text messages for displays and prompts
UNDR	BF98	LB	Label indicator for Single-Step stack underflow
USDB	D85C	DT	Convert to unsigned 8-bit decimal
USDW	D81C	DT	Convert to 16-bit unsigned decimal
USND	D3DB	SS	Prints a line of user's stack contents
USRS	BFC8	WV	Single-Step user's stack pointer for SS display
USST	D3FD	SS	Prints selected line of user's stack
VENT	C8D2	ED	Value entry for getting in various addresses
VERI	C792	CA	VERIFY command handler
VERN	C7FF	EC	Gets in tape name for a 2Ø68 VERIFY
VIDC	C129	OP	Resets the video mode. Unused.

VRVA	D2D4	RC	Enters Single Step from READ (STEP) and waits
WASS	EB31	AS	Main assembly write loop, gets commands, cursor controls
WCMD	CC35	ED	Sets proper return address for EDIT/assembly commands and jumps
WHAR	E298	AS	Determines what register for assembly of bit ops
WHED	CF47	ED	Puts up WRITE heading with END
WHER	DFA2	DS	Looks for a NAME for address in entry buffer (5D24-7)
WHR2	DFA5	DS	Looks for a NAME for address at (HL)
WIND	D06A	SS	Moves in window, executes step, and stores window
WISU	D07C	SC	Clears memory for window display, sets attributes, turns on window (ATTR)
WISW	D2BC	SC	Toggles the window stop
WNAM	CA70	NM	Handles new NAME assignments entered to screen
WNOW	D55A	SS	Selects window/no-window depending on window setup
WOFF	CDC2	ED	Turns off EDIT and returns to READ
WRIT	CDFD	ED	Begins a write to memory in EDIT mode
WRFO	EADD	AS	Advances current write position during mnemonics entry
WTSU	C767	SS	Window transfer set up for exchanging screen files
XREG	D411	SS	Prints exchange registers and their current values
ZADA	C8FA	AS	Calculates addresses for insert and delete, moves affected NAMEs
ZAFP	CC6D	EC	DELETE command handler
ZEND	CCA3	ED	Ends ZAPP routine and restores screen display
ZESC	CCE0	ED	Escape from ZAPP routine when END is too close
ZUFP	CCB6	ED	Handles DELETE when END is less than the cursor address

HOT Z ADDRESSES

HZFG	5C73	BV	Flag byte with 5C73 to control HZ modes; see notes
TEM1	5C9E	WV	Description: First of 9 temporary word storage bins, mostly for relocations
TEM9	5CAE	WV	Last local word variable storage bin
ASIM	5D3Ø	5B	Simulation area for single stepper, which runs ordinary steps here
CSBF	5D8Ø	BF	Buffer for cassette tape header; use data mode
OVER	BF7Ø	LB	Label marker for top of user single step stack
UNDR	BF98	LB	Label indicator for Single-Step stack underflow
FENS	BF9E	WV	Switch address for Single Step window/no-window; starts initialization file
KATT	BFAE	WV	Address of screen-cursor attribute, for setting blink or bright
LFPØ	BFBØ	WV	Address of last floating-point disassembly line for f-p interpreter
BFCO	BFB2	WV	Address of current position in line print buffer
FILC	BFB4	BV	ASCII value of fill character for Fill command
COUN	BFB6	WV	Pointer for printing register display; points to register names
PRIM	BFB8	BV	Holds Ø or 27H for registers or exchange registers; printed
CCOU	BFB9	BV	Delete this one
POIN	BFBA	WV	Single Step's pointer for reading register values
BPT1	BFBC	WV	First breakpoint address
BPT2	BFBE	WV	Second breakpoint address
OSDF	BFCØ	WV	Address of Single-Step display point in window
OSDF	BFC2	WV	Address of Single-Step window's display file
LOSI	BFC4	WV	Last Single-Step instruction address; top line of disassembly
NOSI	BFC6	WV	Next Single-Step instruction address
USRS	BFC8	WV	Single-Step user's stack pointer for SS display
HLEX	BFCA	WV	Single-Step value for HL'; storage from step to step
DEEX	BFCC	WV	Single-Step value for DE'
BCEX	BFCE	WV	Single-Step value for BC'
AFEX	BFDØ	WV	Single-Step value for AF'
IYRG	BFD2	WV	Single-Step value for IY
IXRG	BFD4	WV	Single-Step value for IX
HLRG	BFD6	WV	Single-Step value for HL
DERG	BFD8	WV	Single-Step value for DE
BCRG	BFDA	WV	Single-Step value for BC
AFRG	BFDC	WV	Single-step value for AF
SFBI	BFDE	WV	Storage bin for stack pointer during Single Step
ALN2	BFEØ	WV	Third address slot for alternate NAME file parameters
ALNA	BFE2	4B	Write-in slot for alternate NAME file parameters, two addresses
LENI	BFE6	BV	Length of current instruction during assembly, in bytes
ENDA	BFEA	WV	Current address in END
KADD	BFEE	WV	Address on screen next to cursor, from ARED or KRED
KPOS	BFFØ	BV	Print position on screen for cursor
KLIN	BFF1	BV	Screen line number of line with cursor, for cursor controls
FCBQ	BFF3	BV	Holds displacement byte for indexed bit operations
CBFL	BFF5	BV	Byte flag for disassembly of CB instructions
NTOF	BFF6	WV	Address of low expanding edge of NAME file
NASW	BFF8	WV	NAME file switch address; off if = NEND, on if = NTOP
NADD	BFFC	WV	Next address for disassembly
CADR	BFFE	WV	Current address for disassembly
LD81	CØØØ	EC	Loads ZX tapes to addresses from cursor to END
INBY	CØ1Ø	CA	Gets in one byte from ZX tape
BOUT	CØ63	CA	Break out routine from LD81
TEND	CØ86	CA	Tape-end check routine for LD81
GOBF	CØ96	SC	Main run-to-breakpont routine; saves window setting, forces no window

DBLE	C11E	EC	Resets stack when hexedit cursor is called from hexedit
VIDC	C129	OF	Resets the video mode. Unused.
GTOS	C15E	AS	Finds top of active screen area for assembly or single-step screen
SCFP	C172	AS	Gets scroll line for assembly or single step screen
CKSS	C181	AS	Gets top line of active screen for assembly or single step screens
CASC	C194	CA	Call to EXROM for 2Ø68 cassette routines
SWOU	C1BØ	SC	Switch out window; single step OUT command
PSCR	C1B9	EC	Part-screen command; gets address and disassembles to bottom of screen
PDIS	C1E6	DS	Prints disassembly to screen bottom, restores cursor
PRWS	C1EF	EC	Does a print-screen to 2Ø4Ø in EDIT mode
IMAR	C1F4	SS	Sets cursor for single-step register value entry
LTDF	C1FA	OF	Converts screen line number to display file address in HL
BKAR	C2ØA	AS	Back arrow for assembly editor
FDAR	C211	AS	Forward arrow for assembly editor
KBRI	C218	OF	Sets bright cursor
ATPO	C21E	OF	Sets screen print position corresponding to cursor attribute byte
PFAG	C23B	DS	Sets up to print screen from last cursor address
NAPA	C243	DS	Prints NAMEs in column 14 of disassembly
BFKL	C252	OF	Kills contents of current line print buffer at 5DØ2
SWPM	C26Ø	ED	Sets up parameters for entry in EDIT
IRED	C272	ED	Reads Z8Ø instruction from hex digits on screen
ABRD	C276	OF	Reads from screen starting from column zero
ASCD	C289	AS	Tests for ASCII hex digit (Ø-F), returns C set if not
TLSC	C295	OF	Sets print position for top left of screen
5BIN	C29F	OF	Sends 5-byte string at (HL) to current screen position
RESK	C2A4	OF	Resets the address cursor at top left
AKIN	C2AD	OF	Sets 4 bytes at (HL) into top left corner
$TIN	C2BØ	OF	Sends BC characters at (HL) to current screen position
CLEN	C2CØ	OF	Clears an invalid NAME from screen
B4SP	C2CD	OF	Backs print position 4 spaces for repeat address entry
SDFC	C2D6	OF	Sets DF_CC from current S_POSN, returns S_POSN in HL
SPAC	C2EC	OF	Prints space to line buffer
COMP	C2FØ	OF	Prints comma to line buffer
OPAR	C2F4	OF	Prints open paren to line buffer
CPAR	C2F8	OF	Prints closing parens to line buffer
PRNA	C2FC	OF	Prints 'A' to line buffer
CHAR	C3Ø1	OF	Sends character in A to line buffer; preserves registers
TOPK	C312	OF	Sets main ADDR cursor at top left
ALKE	C318	OF	Waits for a keypress, returns with key in A and C, B=ØØ
NOBA	C325	OF	Searches stack for BASICs return address and jumps to it
BFCL	C32F	OF	Clears line buffer and prints disassembly screen
RDHX	C346	OF	Reads a hex digit from the screen at BC, returns it in A
RSPA	C356	OF	Moves print position 1 space right with wrap around
NESC	C365	NM	Escape routine during NAME entry (when ENTER is hit)
SPPO	C37Ø	OF	Sets current position in line buffer to value in C
ATOH	C37B	OF	Converts ASCII in A to hex
SSPO	C381	OF	Set screen print position from line and column in BC
PRAT	C38D	OF	Set print parameters at line and column in BC
FIAT	C396	OF	Find attribute address for current print position; set DE to input buffer
NCUR	C3AE	NM	Set NAME cursor
ENTP	C3D4	OF	Entry point for END, DEST, LOOK at top left

ENTN	C3B7	OF	Entry point for NAME entry
ENCN	C3BD	OF	Loop for entry of characters of a NAME or address
ADEN	C3C8	OF	Address entry point, test for NAME or hex
NARO	C3DE	NM	Erase/backspace handler for NAME entry
RANA	C3FØ	EC	Reassigns NAMEs to a diplaced area of memory (CHR$ command)
FAPS	C412	RC	PAPER color change command handler
INKS	C426	RC	INK color change command handler
EXFA	C462	AS	Decodes mnemonics of EX (SP) instructions
GLIN	C488	OF	Selects a single line of data or disassembly to fill screen
BTOS	C4DC	OF	Sends line buffer to screen
STAR	C4FØ	OF	Initialize and start up HOT Z
TRAN	C537	EC	TRANSFER command, copies memory contents to DEST
TRNA	C53D	EC	Copies memory and moves NAMEs to DEST; MERGE command in EDIT
NAMV	C564	NM	Main routine for moving NAMEs
MVNM	C576	NM	Suroutine to move a NAME
CLOS	C5A9	NM	Closes gap in NAME file after a move
MNUP	C5CE	NM	Moves part of NAME file up to reorder NAMEs
MVNA	C5E8	NM	Computes addresses for moving NAMEs
GDSP	C614	ED	Gets in displacement for readdressing commands
RADD	C62D	EC	Command handler to readdress a jump table (STR$)
RELO	C65Ø	EC	Command handler for relocator (MOVE)
SV68	C721	EC	2Ø68 SAVE command handler
LD68	C731	EC	2Ø68 LOAD command handler
CNAM	C73A	CA	Gets a tape name for cassette ops
SSWA	C73E	SS	Code to be copied into single step work area for code simulation
KLOD	C76Ø	OF	Loads character in A into a cursor
WTSU	C767	SS	Window transfer set up for exchanging screen files
CASO	C772	CA	Writes tape parameters to cassette buffer (5D8Ø)
VERI	C792	CA	VERIFY command handler
CASN	C79B	CA	Prompts for cassette name and puts it into cassette header buffer
LODN	C7F5	EC	Gets in tape name for a 2Ø68 LOAD
SAVN	C7FA	EC	Gets in tape name for a 2Ø68 SAVE
VERN	C7FF	EC	Gets in tape name for a 2Ø68 VERIFY
HARI	C8Ø4	EC	Hex arithmetic command; prints END + cursor addr, END - Cursor addr
NENT	C83B	EC	Gets in a NAME in data mode, jumps if disassembly NAME assignment
GOMO	C872	SS	Continues run-to-breakpoint, gets or skips window screen
STWD	C89Ø	SS	Stops window if flag set; restores screen after window
GDEC	C8BØ	RC	Gets in decimal address in READ mode
INSS	C8C6	AS	Finds line if no insert, else jumps to insert routine, for code entry
VENT	C8D2	ED	Value entry for getting in various addresses
INAJ	C8EØ	AS	Saves various registers while calling routine to set up insert addresses
ZADA	C8FA	AS	Calculates addresses for insert and delete, moves affected NAMEs
IADA	C91C	AS	Calculates addresses and moves NAMEs for an insert
DSKR	C939	AS	Gets top address on screen, sets print parameters for a down scroll
FIAG	C946	EC	Find again command (ABS) to locate subsequent matches to string
MATS	C954	EC	FIND command handler (SGN)
MAT?	C96F	ED	Tests for match with search string
HUNT	C977	ED	Main search routine for FIND
NFOU	C9AB	ED	Not-found escape from HUNT
MFOU	C9BØ	ED	Match-found escape from HUNT; displays matching location
TOFN	C9E5	RC	Displays beginning of NAME list (RND command)

SHWT	C9EC	ED	Displays new screen starting at HL
DLIS	C9F4	EC	Sends lines to 2Ø4Ø printer from cursor to END (LLIST)
RUNT	CA37	EC	RUN command handler; transfers control to code at cursor
SWNA	CA43	NM	NAME switch (OVER command) to change label files
PRSC	CA5F	RC	Print-screen command for Read and Single Step
CHNA	CA65	NM	Sets values to change and existing label
ANNA	CA6B	NM	Gets ready for another NAME after rejecting one
WNAM	CA7Ø	NM	Handles new NAME assignments entered to screen
CHPT	CAA7	NM	Entry point to WNAM when a NAME already exists for that address
DENA	CAC6	NM	Delete-NAME command handler (EXP)
NTUP	CBØ4	NM	Moves NTOP up after a NAME deletion
NTDN	CBØ9	NM	Moves NTOP down for a NAME to be added
SWDD	CB17	EC	Switches disassembly/data displays during EDIT (THEN)
SUTR	CB4B	ED	Sets up transfer parameters, gets DEST
MOVE	CB6B	ED	Subroutine for transfers, moves code in proper direction
FLMM	CBA1	EC	Fill memory command handler (FN)
CLMM	CBB4	EC	ERASE command handler, fills cursor to END with ØØ
GTOP	CBD3	OP	Gets top of current working screen & renews to bottom
SINS	CBFØ	ED	Sets insert flag and checks for valid END
FILP	CBFC	OP	Fills screen after an editing operation
FILI	CCØ2	ED	Finds current edit line
REDO	CC15	ED	Redoes the screen after edit operations
FPFL	CC28	DS	Sets flags indicating floating-point disassembly in progress
WCMD	CC35	ED	Sets proper return address for EDIT/assembly commands and jumps
SWTE	CC4A	EC	Switch-to-edit command (>=)
HELO	CC64	OP	Gives beep on start up or error return
ZAPP	CC6D	EC	DELETE command handler
ZEND	CCA3	ED	Ends ZAPP routine and restores screen display
ZUPP	CCB6	ED	Handles DELETE when END is less than the cursor address
IESC	CCDB	ED	Escape from insert when instruction will not fit
ZESC	CCEØ	ED	Escape from ZAPP routine when END is too close
GNGO	CCF9	ED	Go/no-go routine for Transfer, Find, etc.
CKIN	CDØB	ED	Checks insert flag and excutes insertion
PUDN	CD39	ED	Pushes down memory contents to make room for insert
CEOP	CD55	ED	Check whether END address is with 256 of cursor and if not ask for new value
SEOP	CD66	ED	Handles TO command to set END
PEOP	CD7C	ED	Prints END value on screen when EDIT cursor is on
OSOU	CDA3	SS	Single Step exit, returns to READ mode
ECMD	CDAA	ED	Calculates offset into jump table for EDIT commands
EDMD	CDB5	RC	Turns on EDIT mode, changes headings, sets cursor
EDRT	CDB8	ED	EDIT command return address
EDCO	CDBB	ED	EDIT command point, waiting for key entry
WOFF	CDC2	ED	Turns off EDIT and returns to READ
KMVS	CDCA	ED	Handles cursor moves during EDIT
ENDE	CDD7	ED	Ends a line edit, moves down cursor, reenters loop
INCK	CDDC	ED	Checks for insert key (EDIT)
KRGT	CDE5	ED	Checks for cursor right, then DELETE, then command keys
WRIT	CDFD	ED	Begins a write to memory in EDIT mode
GOTI	CE2B	ED	Installs code into memory after EDIT entry
EDAT	CE43	ED	Data mode edit routine
EDES	CE7F	ED	Escapes from the middle of an edit entry via ENTER

EDBK	CE88	ED	Backs cursor during edit
POKI	CE93	ED	Inserts code into memory at proper address
MLIN	CEAE	OP	Remakes one line when new instruction is same size as old
EBAK	CEBA	ED	Backs blink bit for cursor left, escapes if too far
ININ	CEC4	ED	Gets in hex code instruction to screen
ADVK	CEE1	ED	Advances edit cursor to the left
FINS	CF12	ED	Tests first instruction for type and length
DUMP	CF40	UU	Dumps all register values to Single Step; a users' utility
WHED	CF47	ED	Puts up WRITE heading with END
KDWN	CF57	ED	Cursor down routine
NEWK	CF62	ED	Calculates new cursor position from HL + DE and sets it
K-UP	CF6C	ED	Handles cursor up commands
DCKS	CF84	ED	Redoes Data display after backing up one address
SCDN	CF9E	ED	Scrolls screen down and finds an instruction to fill the line
SKRL	CFCC	RC	Handles scroll (<>) command; scrolls until BREAK
SCUP	CFE5	ED	Moves screen up for cursor at bottom
SPRD	D001	RC	Reads machine stack pointer and prints it upper right
GUPA	D015	ED	Gets address at line 22 for screen-up
SWFP	D026	RC	Switch floating point interpreter; PEEK command
KOUT	D031	ED	Turns out cursor
IOFF	D037	ED	Switches off insert flag when cursor is moved
KRES	D048	ED	Restores cursor at former position after a command
EDIT	D04D	ED	Sets up cursor at first hexedit position
K-ON	D051	ED	Turns on a top-line cursor at left
KLOC	D054	ED	Records cursor-line as top scren line
OKLO	D056	ED	Records cursor line from A and attribute byte from DE
KURS	D05D	ED	Records cursor attribute byte and sets blink/bright
SKUR	D060	ED	Sets cursor blink and bright if caps shift untoggled
WIND	D06A	SS	Moves in window, executes step, and stores window
WISU	D07C	SC	Clears memory for window display, sets attributes, turns on window (ATTR)
SWIN	D0A3	SC	Checks if there has been a window, switches it IN if so
STTL	D0C1	OP	Clears line buffer and sets screen position to top left
FPAG	D0C7	DS	Finishes a disassembly screen to bottom
SPON	D0CE	RC	Toggles flag to enable or disable SP display (AT)
FPSW	D0D7	RC	Toggles f-p interpreter mode (CODE)
RTBP	D0ED	SS	Runs steps and checks whether a breakpoint has been reached
PPIY	D111	SS	Subroutine for simulation of POP/PUSH IX/IY
SHBP	D122	SS	Displays current breakpoints (AND)
SBP1	D14D	SC	Set breakpoint 1 (AT)
SBP2	D159	SC	Set breakpoint 2 (OR)
SETF	D165	SS	Sets flags register values in register display
OSRS	D193	SC	Set register values command handler (VAL)
SFLA	D1BF	SS	Gets in new setting for flags register
SDRG	D1EC	SS	Sets new value for SP (USRS) in register display
NWRV	D1F6	SS	Installs new register value in register display
GHDG	D22A	ED	Gets in a hex digit, rejects other characters
GALF	D233	ED	Gets in an alphanumeric character, rejects others
GOSV	D245	SS	Handles value entries on Single Step screen
SVAR	D24F	ED	Handles value entries at top left of screen
CRST	D261	SS	Handles RUN CALL command for RSTs
RCAL	D271	SC	RUN CALL command handler (INT)

OSAS	D29E	SC	Enables assembler loop from Single Step (STOP)
WISW	D2BC	SC	Toggles the window stop
OSCO	D2D0	EC	Enters Single Step from EDIT (STEP) and runs step at cursor
VRVA	D2D4	RC	Enters Single Step from READ (STEP) and waits
HOLD	D2D7	SS	Wait point for Single Step command entry
SKIP	D2E9	SS	Handles space key to skip one step
GBAK	D301	SS	Handles EDIT key to back up one step or byte
TERM	D314	SS	Exits from Single Step to READ mode
OSNA	D31F	SS	Handles NAME entry to Single Step screen
OSCM	D33B	SS	Processes Single Step commands
RDIS	D348	SS	Prints Single Step screen
RESC	D39F	SS	Resets lower part of screen only
STAK	D3CC	SS	Prints current user's stack on single step screen
USND	D3DB	SS	Prints a line of user's stack contents
ADNA	D3EB	SS	Prints address, three spaces, and corresponding NAME if any
USST	D3FD	SS	Prints selected line of user's stack
XREG	D411	SS	Prints exchange registers and their current values
MREG	D41D	SS	Prints main registers and their current values
ALIN	D483	SS	Prints A register line in Single Step display
APRI	D4BA	SS	Prints A'register line for Single Step display
EXFL	D4C4	SS	Prints exchange flag value to Single Step screen
FLAG	D4CC	SS	Prints flag values to Single Step screen
ARE3	D512	OP	Reads and address from screen and preserves BC
PUDS	D518	DT	Prints unsigned decimal byte and a space
PEXP	D51F	SS	Prints 'EXFLAGS'
PFLA	D524	SS	Prints 'FLAGS'
PSSP	D52C	SS	Prints 'SP '
ONES	D53A	SS	Reads EDIT cursor and runs the instruction there
STEP	D53D	SS	Sets up simulation area and runs current instruction as a step
STE2	D540	SS	Steps current instruction in NOSI
WNOW	D55A	SS	Selects window/no-window depending on window setup
SSOR	D55E	SS	Sorts step instruction, selects simulation routine if needed
SORT	D570	SS	Sorts for simulation type of step instruction
SRST	D600	SS	Simulation routine for stepping RSTs
SINC	D616	SS	Simulation routine for stepping INC/DEC SP
SSPH	D627	SS	Simulation routine for stepping LD SP,HL
SIJP	D62D	SS	Simulation routine for stepping JPs
SICA	D647	SS	Simulation routine for stepping CALLs
BORS	D66A	RC	BORDER color set command (BRIGHT)
SPUP	D670	SS	Simulation routine for stepping PUSH/POP
SRET	D67F	SS	Simulation routine for stepping RETs
SSPL	D69B	SS	Simulation routine for stepping LD SP,NNNN
SSPT	D6A9	SS	Simulation routine for stepping LD SP,(NNNN)
SJPH	D6B4	SS	Simulation routine for stepping JP(HL/IX/IY)
SIJR	D6D8	SS	Simulation routine for stepping JRs
SUWA	D704	SS	Sets up stepper work area in printer buffer
CRUN	D714	SS	Loads all registers, runs step, saves all registers
SDON	D733	SS	Entry to CRUN for some simulation routines
DUM2	D737	UU	Entry point for DUMP utility
FINI	D74A	SS	Entry to CRUN for some simulation routines
DISA	D759	DS	Main disassembler loop

CHGD	D764	DS	Changes display between Data and Disassembly
PTOP	D76C	DS	Reprints display from top (line 2)
OSRT	D771	SS	Single-Step return point to READ
CHOO	D774	DS	Selects Data/Disassembly according to flag bit 4
DATP	D77E	DT	Prints full screen of data display
DATL	D787	DT	Prints one line of data display
DDAT	D7AD	DT	Main routine for printing data display
PCH$	D7C4	DT	Looks up BASIC tokens and prints them
DIVI	D7E1	DT	Divides HL by BC for decimal conversions
NSGN	D7ED	DT	Handles alignment of minus sign on negative decimals
JUST	D8Ø2	DT	Right justifies decimal numbers
PLAD	D6ØF	DT	Prints last digit of decimal number
SDWO	D817	DI	Convert to 16-bit signed decimal
USDW	D81C	DT	Convert to 16-bit unsigned decimal
PSDW	D83B	DT	Print 16-bit signed decimal
PUDW	D64Ø	DT	Print 16-bit unsigned decimal
PSDB	D846	DT	Print 8-bit signed decimal (-128 to 127)
PUDB	D84F	DT	Print 8-bit unsigned decimal (Ø to 255)
SDBY	D854	DT	Convert to signed 8-bit decimal
USDB	D85C	DT	Convert to unsigned 8-bit decimal
PDAD	D865	DT	Print decimal address column for data display
LNAM	D87E	DT	Print NAME column for Data display
FLCK	D898	DS	Checks disassembly flags for RST 28 or RST Ø8 in progress
FLOT	D8AD	DS	Begins f-p interpreter for RST 28
PFPC	D8C1	DS	Prints floating point column in disassembly
RSFP	D91A	DS	Reads floating point data from code stream
LFPD	D91F	DS	Lists floating point data as decimal
FPDA	D958	DS	Main floating-point data interpreter
PBOT	D96F	DS	Records last floating point operation in LFPO at bottom of screen
COFP	D97B	DS	Interprets f-p constant-to-stack operators
PFPO	D9AØ	DS	Looks up and prints floating point operator mnemonics
FTOB	D9AØ	DS	Moves characters from mnemonics file to line buffer
FPJR	D9AB	DS	Interprets f-p op relative jumps
RDRS	D9BB	DS	Return point for RST 28, Ø8 disassembly routines
BERR	D9C7	DS	Prints ERROR after RST Ø8 and checks report number
CHA2	D9D5	DS	Converts hex value to ASCII and sends it to line print buffer
SKID	D9DB	DS	Skips over ordinary disassembly for RST Ø8, 28 ops
OLIN	D9ED	DS	Disassembles a single instruction and prints line
DPAG	DAØB	DS	Disassembles and lists to end of screen
RSTD	DA26	DS	Disassembles RST instructions
CONL	DA58	DS	Disassembles conditional forms, Z, NZ, etc.
NUMB	DA65	DS	Gets 8-bit hex digit to disassembly for direct loads, etc.
ACON	DA7C	DS	Prints (NNNN) forms in disassembly
LURP	DA86	DS	Look up register pair for disassembly
ADJR	DAA7	DS	Calculates destination address for JRs
BKWD	DAB3	DS	Calculates destination address for backward JRs
HLIX	DABB	DS	Distinguishes HL, IX, and IY in disassembly
PCON	DAC7	DS	Prints forms (HL), (IX), (IY) for disassembly
DADR	DADØ	DS	Prints 16-bit number or address NAME for disassembly
MNAM	DAD8	DS	Looks up NAME and prints it or address if none
HEAD	DAE4	DS	Prints READ mode column headings

NCOL	DAF9	DS	Prints disassembly mode NAME column
BLAN	DAFF	DS	Prints blank if no NAME, else one space
NAME	DBØ8	DS	Looks up NAME at CADR and prints if there is one
FIND	DB1C	DS	Searches NAME file for a NAME at CADR
2FIN	DB1F	DS	Searches file for a NAME at (HL)
3FIN	DB2Ø	DS	Searches file for a NAME at (DE)
FINA	DB3C	DS	NAME file search routine, expects address in DE else enter at F2NA
CPBC	DB8A	DS	Compares BC and DE, returns Z for match, NC if DE larger
SHLP	DB92	DF	Set HL' to proper value for return to ROM
DISØ	DB98	DS	Disassemble op codes from ØØ to 3F
DIS3	DB9D	DS	Disassemble op codes from CØ to FF
DISS	DBAØ	DS	Main disassembly loop
IXIY	DBDD	DS	Set FDDØ flag for DD or FD prefixes
LOOK	DBE9	DS	Sorts instructions for disassembly look up
DIS2	DCØD	DS	Disassemble op codes from 8Ø to BF
CBDI	DC2D	DS	Disassemble bit ops (codes with CB prefix)
ROIN	DC69	DS	Disassemble rotate and shift instructions
DIS1	DC78	DS	Disassemble op codes from 4Ø to 7F (8-bit LDs)
FREG	DC89	DS	Adds final register to 8-bit register LDs
CREG	DC8E	DS	Identifies first register in 8-bit LDs
TREG	DC97	DS	Identifies second register in LDs or register in arithmetic ops
MNAR	DCC7	DS	Takes argument from mnemonics file and jumps to handler routine
ED1I	DCD3	DS	Disassembles op codes ED4Ø to ED7F
MNEM	DDØ5	DS	Reads mnemonics from file to line buffer, traps argument byte
EDDI	DD3E	DS	Sorts ED prefixed ops for disassembly
PDAT	DD4C	DS	Prints DATA for invalid ED instructions
PHLT	DD51	DS	Prints HALT mnemonic
PRLD	DD59	DS	Prints LD
PERR	DD61	DS	Prints ERROR for RST Ø8 instruction sequel
ED3I	DD6B	DS	Disassembles op codes from ED8Ø to EDBF
MNPR	DD87	DS	Finds mnemonic in file and sends it to (DE)
PFIL	DD8A	DS	Prints from any file with bit 7 set for last character
INYE	DD9C	DS	Prints indexed displacement from IY for unNAMEd slots among SVs
INSY	DDA4	DS	Gets system variable NAMEs for indexed displacement reference
INDX	DDC8	DS	Gets displacement for indexed forms, prints, closes parens
DISP	DDD1	DS	Sorts direct loads to IX/IY from indexed displacements
LDIN	DDDE	DS	Handles direct loads to IX/IY
OCEX	DDFØ	DS	Exchanges 2nd and 3rd octal digits of a hex byte
COCT	DDF9	DS	Gets second octal digit of A into A
AVCA	DEØ2	DS	Advance current disassembly address
MFIN	DEØA	DS	Finds numbered entry in A in table at (HL), end bits 7 set
SCND	DE16	DS	Tests screen bottom, returns NC if last line printed
CLMN	DE1F	DS	Clears old mnemonic from display screen prior to printing current one
ADVA	DE33	DS	Advance current address to next instruction address
CODE	DE3A	DS	Get instruction length and print hexcode column
ILEN	DE62	DS	Look up length for instruction byte at (HL)
REAC	DECA	DS	Reactivate ADDR cursor after invalid NAME entry
DSCO	DEDD	DS	READ mode command point, waiting for entry
RCMD	DEF5	DS	Sets return address for READ commands, looks up and jumps to command
ENTR	DFØB	DS	Looks up address/NAME entries
EADR	DF1C	DS	Reads entered address from ADDR slot at top left

RIAD	DF23	DS	Reads a hex address from screen at top left
RIA2	DF24	DS	Reads a hex address at left of line in A
KRED	DF33	OF	Reads address at left of cursor line
ARE2	DF36	DS	Reads address at left of line in A
ENNA	DF58	DS	Entry loop for NAME at top left
OSEN	DF6C	SS	Entry loop for NAME at line 18 in Single Step
JCMD	DF89	EC	Sets table base for edit command jump table
DON?	DF8E	NM	Checks whether a NAME look up is completed
DSWI	DF9E	RC	Data/disassembly display switch, THEN command in READ
WHER	DFA2	DS	Looks for a NAME for address in entry buffer (5D24-7)
WHR2	DFA5	DS	Looks for a NAME for address at (HL)
JPHL	DFCA	OF	Jumps to Ath address in jump table at (HL)
ADDL	DFD3	OF	Performs HL = HL + A, preserves A
HWOR	DFDB	DS	Sends current disassembly address to line buffer; address column
DEWD	DFDF	DS	Sends hex number in DE to line buffer for printing
HBYT	DFE4	DS	Sends hex byte in A to line buffer for printing
FNIB	DFF1	DS	Sends one nibble of hex byte to line buffer for printing
SCDE	DFFA	OF	Sends hex number in DE direct to screen
HTOA	EØØ7	OF	Converts hex to ASCII in A
LINE	EØ1Ø	OF	Sets screen position to BC and draws 32 character line across
CLLI	EØ18	OF	Fills line print buffer with 32 spaces (2ØH)
ANYC	EØ1E	OF	Sends character in C to line print buffer B times
DSPA	EØ25	OF	Prints double space
FRI$	EØ2A	OF	Sends character string of length BC at (HL) to line print buffer
IFCH	EØ3B	OF	Filters non-print characters before sending to line print buffer
4CHR	EØ41	OF	Sends a 4-charcter string to line print buffer
OHED	EØ48	SS	Prints Single-Step screen column headings
DHED	EØ4D	DS	Prints disassembly screen column headings
KHED	EØ52	DT	Prints data screen column headings
CLWA	EØ67	OF	Clears BASIC's work area to remove old address entries
KEYB	EØ6E	OF	Keyboard read; waits for a key, beeps, controls modes
ACKN	EØ9B	OF	Acknowledges valid keystrokes with beep
OSBS	EØB2	SS	Handles value setting for A and F registers in step mode
FINS	EØC3	ED	Gets length of instruction at insert cursor
MDB"	EØD2	AS	Handles assembly op DB when used with quoted string
M-DB	EØE9	AS	Assembles DB when used with hex numbers (no quotes)
DBEN	E121	AS	Puts DB bytes into memory and redoes screen to hide them
MHAL	E149	AS	Assembles HALT instruction
M-IM	E153	AS	Assembles IM instructions
ETRK	E171	AS	Assembly local error trap
M-EX	E174	AS	Assembles EX instructions
MRST	E1A4	AS	Assembles RST instructions
IRDR	E1B5	AS	Subroutine for assembly of LDI, LDIR, LDDR, LDD and similar instructions
MCFI	E1CD	AS	Assembles CPI, CPIR
MINI	E1D1	AS	Assembles INI, INIR
MLDD	E1D5	AS	Assembles LDD, LDDR
MCFD	E1D9	AS	Assembles CPD, CPDR
MLDI	E1D9	AS	Assembles LDI, LDIR
MIND	E1DD	AS	Assembles IND, INDR
MOTI	E1E1	AS	Assembles OTIR
MOTD	E1E5	AS	Assembles OTDR

MOUT	E1EE	AS	Assembles OUT, OUTI, OUTD instructions
M-IN	E224	AS	Assembles IN instructions
ETRJ	E244	AS	Local assembly error trap
IORG	E247	AS	Gets I/O register for IN X,(C) or OUT (C),X
CORN	E25B	AS	Gets (C) or (NN) for assembly of INs and OUTs
MBIT	E27D	AS	Assembles BIT instructions
MRES	E281	AS	Assembles RES instructions
MSET	E285	AS	Assembles SET instructions
BIMN	E287	AS	Subroutine for assembly of BIT, RES, SET
WHAR	E298	AS	Determines what register for assembly of bit ops
BOIX	E2A6	AS	Assembles indexed bit ops
MRLC	E2AF	AS	Assembles RLC instructions
MRRC	E2B5	AS	Assembles RRC instructions
M-RL	E2C6	AS	Assembles RL instructions
M-RR	E2CA	AS	Assembles RR instructions
MSLA	E2D3	AS	Assembles SLA instructions
MSRA	E2D7	AS	Assembles SRA instructions
MSRL	E2DB	AS	Assembles SRL instructions
MRET	E2E2	AS	Assembles RET instructions
ETRI	E315	AS	Local assembly error trap
MPOP	E318	AS	Assembles POP instructions
MPUS	E31F	AS	Assembles PUSH instructions
PPIX	E33F	AS	Subroutine for PUSH/POP of IX/IY
MINC	E34E	AS	Assemble INC commands
MDEC	E352	AS	Assemble DEC instructions
ID81	E371	AS	Codes for INC/DEC (IX/Y+NN)
ID16	E378	AS	Identifies 16-bit register pair for INC/DEC instructions
IDIX	E393	AS	Codes for INC/DEC IX/Y
AR16	E3AE	AS	Codes for 16-bit ADC, SBC, ADD
ACSH	E3E5	AS	Codes for 16-bit ADD
MSUB	E42B	AS	Assembles SUB instructions
MAND	E42C	AS	Assembles AND instructions
MXOR	E43Ø	AS	Assembles XOR instructions
MADD	E437	AS	Assembles ADD instructions
MADC	E43B	AS	Assembles ADC instructions
MSBC	E43F	AS	Assembles SBC instructions
8AOP	E44D	AS	Codes for 8-bit arithmetic ops
M-OR	E46Ø	AS	Assembles OR instructions
M-CP	E464	AS	Assembles CP instructions
MDAO	E47Ø	AS	Codes for direct arithmetic ops (e.g., ADD A,NN)
FWDJ	E48E	AS	Codes for JRs entered with +N for displacement
FWCJ	E49Ø	AS	Codes for conditional JRs entered with +N displacement
MJRS	E4AE	AS	Assembles JR instructions
MDJN	E4E7	AS	Assembles DJNZ
JRDI	E4F4	AS	Calculates displacement for relative jumps
MJPC	E52C	AS	Assembles JP instructions
MCAL	E56Ø	AS	Assembles CALL instructions
CAJP	E56D	AS	Subroutine for assembly of CALLs and JPs
MALD	E582	AS	Assembles LD instructions; sorts on comma position
MLD8	E5C9	AS	Codes for LD (IY+NN),XX,where XX is register or number
DIRL	E5FF	AS	Codes for direct index register LD

MLD7	E6ØB	AS	Codes for LD (ADDR),RR
MLD5	E61C	AS	Codes for LD (RR),X
DLHL	E656	AS	Codes for LD (HL),N
ATBC	E65F	AS	Codes for LD (BC),A
ATDE	E665	AS	Codes for LD (DE),A
MLD3	E679	AS	Codes for LD RR,XX, where XX is direct or indirect expression or HL
DDLD	E6B7	AS	Codes for LD RR,NNNN (direct double load)
IDLD	E6D4	AS	Codes for LD RR,(ADDR) (indirect double load)
MLD2	E7Ø8	AS	Codes for LD R,X, where X is any option
INRE	E73D	AS	Codes for LDs to and from I and R registers
ML29	E759	AS	Codes for LD R,(IX+NN), where R is an 8-bit register
ML28	E766	AS	Codes for LD A,(ADDR)
ML26	E773	AS	Codes for LD R,(RR), where RR is HL, IX/Y, DE, BC
ML25	E7AA	AS	Codes for LD R,'A', where A is ASCII character
GQUO	E7B5	AS	Checks quotes and gets character into E
ML24	E7C1	AS	Codes for direct 8-bit register loads, LD R,NN
ML23	E7CF	AS	Codes for 8-bit register to register loads, LD R,R'
REIN	E7E3	AS	Codes for LD A,I and LD A,R
LDAD	E7F7	AS	Codes for an address when assembling LDs
NORA	E7FA	AS	Assembler check routien for NAME or address
RGXF	E83E	AS	Checks syntax and gets displacement for IX+NN forms
RGX8	E842	AS	Identifies 8-bit registers for coding
RDBL	E87Ø	AS	Identifies 16-bit register pair for coding
IX+N	E86A	AS	Checks and codes for (IX+NN) forms
TIXY	E88D	AS	Checks and codes for IX+NN)
ADFN	E8A7	AS	Gets numeric address for a NAME
EVAD	E8C1	AS	Evaluates address (ADDR) for assembly
CMFO	E8DE	AS	Determines position of comma in a mnemonic entry
SFAF	E8F1	AS	Determines next blank space position in a mnemonic entry
NACK	E8F5	AS	Checks whether a sequence of characters is a NAME
RHEX	E9ØF	AS	Reads a hex byte from mnemonic to E
HIDG	E91C	AS	Shifts high nibble of hex byte left
FIØ8	E925	AS	Gets a number from Ø to 7 or reports error
FIØF	E931	AS	Tests ASCII for hex digit ØØ to ØF and converts to hex
MCND	E93D	AS	Subroutine for assembly of conditional mnemonics
RG16	E961	AS	Looks up 16-bit registers in table
REG8	E97B	AS	Looks up 8-bit registers in table
CKRX	E98F	AS	Checks for an X and returns with Z or NZ
CKRH	E993	AS	Checks for an H and returns Z or NZ
CKR(	E997	AS	Checks for (and returns Z or NZ
CKRA	E99B	AS	Checks for an A and returns Z or NZ
CKRS	E99F	AS	Checks for a space and returns Z or NZ
CIRA	E9A3	AS	Checks mnemonic for an initial A and returns Z or NZ
CIR(	E9A7	AS	Checks mnemonic for initial (and returns Z or NZ
CIRS	E9AB	AS	Checks mnemonic for initial space and returns Z or NZ
CIRU	E9AD	AS	Sets 'initial' position, checks value against A and returns
CKRU	E9B5	AS	Advances position counter, checks value against A, returns
CKTL	E9B9	AS	Check for an L in mnemonic and go to error trap if not
CKTI	E9BD	AS	Check for an I in mnemonic and go to error trap if not
CKT)	E9C1	AS	Check for) in mnemonic and go to error trap if not
CKT+	E9C5	AS	Check for + in mnemonic and go to error trap if not

CKTV	E9C9	AS	Check for a comma in mnemonic and go to error trap if not
CKT(	E9CD	AS	Check for (in mnemonic and go to error trap if not
CKTA	E9D1	AS	Check for A in mnemonic and go to error trap if not
CKTS	E9D5	AS	Check for space in mnemonic and go to error trap if not
CITA	E9D9	AS	Check for 'initial' A in mnemonic and go to error trap if not
CIT(	E9DD	AS	Check for 'initial' (in mnemonic and go to error trap if not
CITS	E9E1	AS	Check for 'initial' space and go to error trap if not
CITU	E9E3	AS	Set 'initial' position and compare with A, trap if not the same
CKTU	E9EB	AS	Compare character in mnemonic with A, trap if not the same
FIMN	E9F2	AS	Finds match in table for the first 3 letters of the mnemonic
SWAS	EA1C	EC	Switch from hexedit to assembly edit (STOP command)
FIXD	EA3C	AS	Assembles invariant mnemonics, e.g., CCF, SCF
RDIT	EA57	AS	Main routine for reading back and assembling mnemonics
INCO	EA6A	AS	Locates entry point for code entry, handles insertions
INDO	EA8E	AS	Resets disassembly after assembled code is entered
RETE	EAA7	AS	Return point for syntax error traps, flags errant character
EROF	EAB1	AS	Continues syntax error processing
CESC	EABF	AS	Escape from assembly when ';' key is pressed
ASRT	EACE	AS	Return point for assembly-edit commands
WRFO	EADD	AS	Advances current write position during mnemonics entry
CINS	EAED	AS	Inserts space at cursor during menmonics entry
CTSC	EAFE	AS	Checks for space or comma; used after conditionals
ERAS	EB07	AS	Deletes character behind cursor during mnemonics entry
ASED	EB20	AS	Entry to assembly edit from READ mode (STOP command)
WASS	EB31	AS	Main assembly write loop, gets commands, cursor controls
MNLO	EB7A	AS	Mnemonics edit loop for entry
STEN	EBA5	AS	Start entry by printing initial character to screen
KSRT	EBA9	AS	Moves line cursor right
EREN	EBAA	AS	Re-entry point after error trap
OKIN	EBAE	AS	Mnemonic is ready; put it in
EERT	EBB1	AS	Address on stack used by syntax error trap
BKSF	EBB7	AS	Backspace during assembly line edit
ACMD	EBC7	AS	Sorts assembly-edit commands
FDSF	EBD1	AS	Right arrow during assembly line edit
COLR	EBDA	RC	Gets in color number for INK, PAPER, BORDER commands
FSTA	EC0D	SS	Prints STACK on register display
FL-N	EC15	SS	Prints LAST-NEXT on register display
SOFF	EC1E	RC	Sign off; installs current NAME file as permanent, goes to BASIC
SEND	EC36	RC	Sets END from READ mode (TO command)
DELE	EC42	AS	Removes a character from screen during assembly edit
OFES	EC5A	AS	Opens space in assembly edit line for insertion
RDUF	EC74	AS	Reads mnemonic entry from screen to buffer at 5D15 for syntax check
SORC	EC99	AS	Searches mnemonic string for first space or comma
FFAT	ECC1	DS	Continues floating point disassembly
HZET	ECE7	UF	Fields the ROM error traps when HOT Z is running
CSUM	ED07	EC	Checksum command (LEN)
DCIN	ED3A	DS	Gets in decimal address for next disassembly page
MNBA	EDAD	FI	File of mnemonics for assembly
FXBA	EE60	FI	File of codes of 'fixed' mnemonics
DBL1	EE74	FI	File of second character of double register names
CNBA	EE7B	FI	File of ASCII conditional particles

HED3	EE8C	FI	Characters for single step column headings
HED2	EEAC	FI	Characters for data display column headings
HED1	EECC	FI	Characters for disassembly column headings
LNFI	EEEC	FI	File of instruction lengths
RG8F	EF6C	FI	File of 8-bit register names
EDFI	EF8Ø	FI	Mnemonics file for disassembly of high ED instructions
DBLF	EFBB	FI	File of double register names
CPFI	EFC5	FI	File of conditionals for disassembler
DTFI	EFD2	FI	Various disassembler text messages
E2FI	EFE3	FI	Disassembler mnemonics for low ED instructions
CBFI	FØ72	FI	File of mnemonics for CB instructions
MNFI	FØ9C	FI	File of mnemonics for main instruction sequence
DBLR	F28A	FI	Double register file for arithmetic ops
CPFI	F297	FI	Conditional particle file for disassembler
FPCF	F334	FI	File of floating-point constants (ASCII)
TXFI	F346	FI	Various text messages for displays and prompts
IVAR	F3CE	FI	File of initial HOT Z system variables for startup
FPFI	F43Ø	FI	File of floating point mnemonics
MNAD	F4F6	JT	Assembler routine jump table
DAFI	F54E	JT	Disassembler mnemonics argument jump table
SIFI	F562	JT	Single Step simulation jump table
CDFI	F57E	JT	Command jump table (Step, Read, Edit, each starting with RND key)

HOT Z-2Ø68 COMMAND LIST -- READ MODE

COMMAND KEY		FUNCTION	ROUTINE
SPACE	SPACE	PAGE flip	
SS-Q	<=	QUIT TO BASIC (SIGN OFF)	SOFF
CSS-COPY	LN	COPY screen to 2Ø4Ø	PRSC
SS-E	>=	Turn on HEXEDIT mode	EDMD
SS-A	STOP	Turn on ASSEMBLY mode	ASED
CSS-T	RND	Display TOP NAME of list	TOPN
CSS-SS-N	OVER	Switch NAME files	SWNA
CSS-R	INT	RESTART HOT Z (Reinitialize)	STAR
CSS-REM	TAN	Make REM from PROG to END	REMK
CSS-SS-BORDR	BRIGHT	Set BORDER color (Ø-7)	BORS
CSS-SS-X	INK	Set INK color (Ø-7)	INKS
CSS-SS-C	PAPER	Set PAPER color (Ø-7)	PAPS
SS-D	STEP	Go to single STEP	VRVA
SS-G	THEN	Switch disassembly/data displays	DSWI
SS-F	TO	Set END address	SEND
SS-U	OR	DECIMAL address to follow	GDEC
SS-W	<>	SCROLL display (BREAK to stop)	SKRL
SS-I	AT	Display machine STACK POINTER (switch)	SPON
CSS-O	PEEK	Switch floating-point interpreter IN/OUT	SWFP
CSS-I	CODE	Switch floating-point INTERPRETATION	FPSW
CSS-H	SQR	HELP screen (v. 1.61 only)	HELP

HOT Z-2Ø68 COMMAND LIST -- SINGLE-STEP MODE

SS-Q	<=	QUIT to READ mode	
ENTER	ENTER	STEP one instruction	
SPACE	SPACE	SKIP next instruction	
CS-1	EDIT	BACK one instruction (or byte if repeated)	
CSS-COPY	LN	COPY to 2Ø4Ø printer	PRSC
CSS-RUN	INT	RUN CALL or RST 1Ø	RCAL
SS-I	AT	Set BREAKPOINT #1	SBP1
SS-U	OR	Set BREAKPOINT #2	SBP2
SS-Y	AND	DISPLAY Breakpoints	SHBP
SS-G	THEN	GO (run) to breakpoint	RTBP
CSS-LOAD	VAL	LOAD register (A,B,D,F,H,S,X,Y)	OSRS
SS-A	STOP	ASSEMBLE NEXT	OSAS
CSS-SS-L	ATTR	Window SETUP at NEXT address (1BØØ bytes)	WISU
CSS-SS-K	SCREEN	Window STOP switch	WISW
CSS-SS-O	OUT	Switch window out temporarily	SWOU
CSS-SS-I	IN	Switch window in again	SWIN
CSS-H	SQR	HELP screen (v. 1.61 only)	HELS

****

STEP command addresses are in a file at CDFI, followed by READ command addresses, followed by EDIT addresses. Dead keys are marked DeAD in STEP and READ and KRES in EDIT. Command addresses are in keycode order from RND through RESTORE, repeating for each mode. Presence of an address assigns that routine to that key. Move them or add to them to suit your needs. Appendix B of 2Ø68 manual gives keycode order.

HOT Z-2Ø68 COMMAND LIST -- EDIT MODE

Command Key		Function	Routine
SS-O	:	ESCAPE during assembly edit	
SS-E	<=	Cursor to HEXEDIT column	SWTE
SS-A	STOP	Move cursor to ASSEMBLY-edit column	SWAS
ENTER	ENTER	ESCAPE during hex edit, or return to READ mode from home column	
SS-D	STEP	Single-STEP instruction at cursor	OSCO
SS-F	TO	Set END	SEOP
CSS-F	SGN	FIND first matching byte sequence	MATS
CSS-G	ABS	FIND NEXT matching byte sequence	FIAG
CSS-N	INKEY$	NAME entry (disassembly or data)	NENT
CSS-X	EXP	DELETE NAME	DENA
CSS-SS-7	ERASE	CLEAR memory from cursor to END	CLMM
CSS-SS-2	FN	FILL memory with keycode	FLMM
CSS-SAVE	RESTORE	SAVE cursor to END in DATA format	SV68
SS-CSS-R	VERIFY	VERIFY a code-format tape	VERI
CSS-LOAD	VAL	LOAD (DATA) from cursor to END	LD68
CSS-W	COS	LOAD ZX81 data tape, cursor to END	LD81
CSS-T	RND	TRANSFER cursor-END to DEST	TRAN
CSS-SS-T	MERGE	TRANSFER code and labels to DEST	TRNA
SS-G	THEN	SWITCH DISPLAY (disassembly/data)	SWDD
CSS-RUN	INT	RUN from cursor to first RET	RUNT
CSS-K	LEN	CHECKSUM to BCDE in single step	CSUM
CSS-V	LLIST	LIST cursor to END on 2Ø4Ø printer	DLIS
CSS-COPY	LN	COPY screen to 2Ø4Ø printer	PRWS
CSS-A	READ	Hex ARITHMETIC (E + K & E - K)	HARI
SS-I	AT	PART screen (enter address)	PSCR
CSS-SS-6	MOVE	RELOCATE code, cursor to END (Set TEMs)	RELO
CSS-Y	STR$	READDRESS jump table (displacement)	RADD
CSS-U	CHR$	READDRESS NAME file (displacement)	RANA
SS-OR	OR	Set END = cursor address	KTOE
CSS-H	SQR	Help screen (v. 1.61)	HELE

INDEX OF COMMAND REFERENCES

READ COMMANDS

			Page
SPACE	SPACE	PAGE flip	4
SS-Q	<=	QUIT TO BASIC	2Ø
CSS-COPY	LN	COPY screen to 2Ø4Ø	2Ø
SS-E	>=	Turn on HEXEDIT mode	2Ø
SS-A	STOP	Turn on ASSEMBLY mode	2Ø
CSS-T	RND	Display TOP NAME of list	2Ø,29,5
CSS-SS-N	OVER	Switch NAME files	2Ø,29,34
CSS-R	INT	RESTART HOT Z	2Ø
CSS-REM	TAN	Make REM from PROG to END	21
CSS-SS-BORDR	BRIGHT	Set BORDER color (Ø-7)	21,4
CSS-SS-X	INK	Set INK color (Ø-7)	21,4
CSS-SS-C	PAPER	Set PAPER color (Ø-7)	21,4
SS-D	STEP	Go to single STEP	21,6
SS-G	THEN	Switch disassembly/data	21,4-5
SS-F	TO	Set END address	21
SS-U	OR	DECIMAL address to follow	21,5
SS-W	<>	SCROLL display	21
SS-I	AT	Display STACK POINTER	21
CSS-O	PEEK	Switch fp interpreter	21,38,5
CSS-I	CODE	Switch INTERPRETATION	22,38,6
CSS-H	SQR	HELP screen (v. 1.61)	8,9

SINGLE-STEP COMMANDS

			Page
SS-Q	<=	QUIT to READ mode	26,8
ENTER	ENTER	STEP one instruction	26,7
SPACE	SPACE	SKIP next instruction	26,7
CS-1	EDIT	BACK one instruction	26
CSS-COPY	LN	COPY to 2Ø4Ø printer	26
CSS-RUN	INT	RUN CALL or RST 1Ø	26
SS-I	AT	Set BREAKPOINT #1	26,7
SS-U	OR	Set BREAKPOINT #2	26,7
SS-Y	AND	DISPLAY Breakpoints	27,7
SS-G	THEN	GO (run) to breakpoint	27,7
CSS-LOAD	VAL	LOAD register	27
SS-A	STOP	ASSEMBLE NEXT	27,8
CSS-SS-L	ATTR	Window SETUP at NEXT	27,18
CSS-SS-K	SCREEN	Window STOP switch	27,19
CSS-SS-O	OUT	Switch window out	27,18
CSS-SS-I	IN	Switch window in again	27,18
CSS-H	SQR	HELP screen (v. 1.61)	8,9

INDEX OF COMMAND REFERENCES

EDIT MODE

			Page
SS-O	;	ESCAPE during assembly	23
SS-E	>=	Cursor to HEXEDIT column	22,12,15
SS-A	STOP	Move cursor to ASSEMBLY	22,1Ø,15
ENTER	ENTER	ESCAPE during hex edit, or return to READ mode	23,1Ø
SS-D	STEP	Single-STEP instruction	23,15
SS-F	TO	Set END	23,14
CSS-F	SGN	FIND first match	23,16
CSS-G	ABS	FIND NEXT match	23,16
CSS-N	INKEY$	NAME entry	23,11,15,28
CSS-X	EXP	DELETE NAME	24,15,29
CSS-SS-7	ERASE	CLEAR memory	24,15,17
CSS-SS-2	FN	FILL memory	24,15,17
CSS-SAVE	RESTORE	SAVE	24,15,16,37
SS-CSS-R	VERIFY	VERIFY	24,16
CSS-LOAD	VAL	LOAD (DATA)	24,15,16
CSS-W	COS	LOAD ZX81 data	24
CSS-T	RND	TRANSFER to DEST	24,15,16,35
CSS-SS-T	MERGE	TRANSFER with labels	24,16,35
SS-G	THEN	SWITCH DISPLAY	25,13
CSS-RUN	INT	RUN to first RET	25,15
CSS-K	LEN	CHECKSUM to BCDE	25
CSS-V	LLIST	LIST on 2Ø4Ø printer	25,15,17
CSS-COPY	LN	COPY screen to 2Ø4Ø	25
CSS-A	READ	Hex ARITHMETIC	25,17,36
SS-I	AT	PART screen	25
CSS-SS-6	MOVE	RELOCATE code	25,15,32,35
CSS-Y	STR$	READDRESS jump table	25,15,32,36
CSS-U	CHR$	READDRESS NAME file	25,15,34
SS-OR	OR	Set END = cursor address	25,14
CSS-H	SQR	Help screen (v. 1.61)	8